Cube Pruning as Heuristic Search

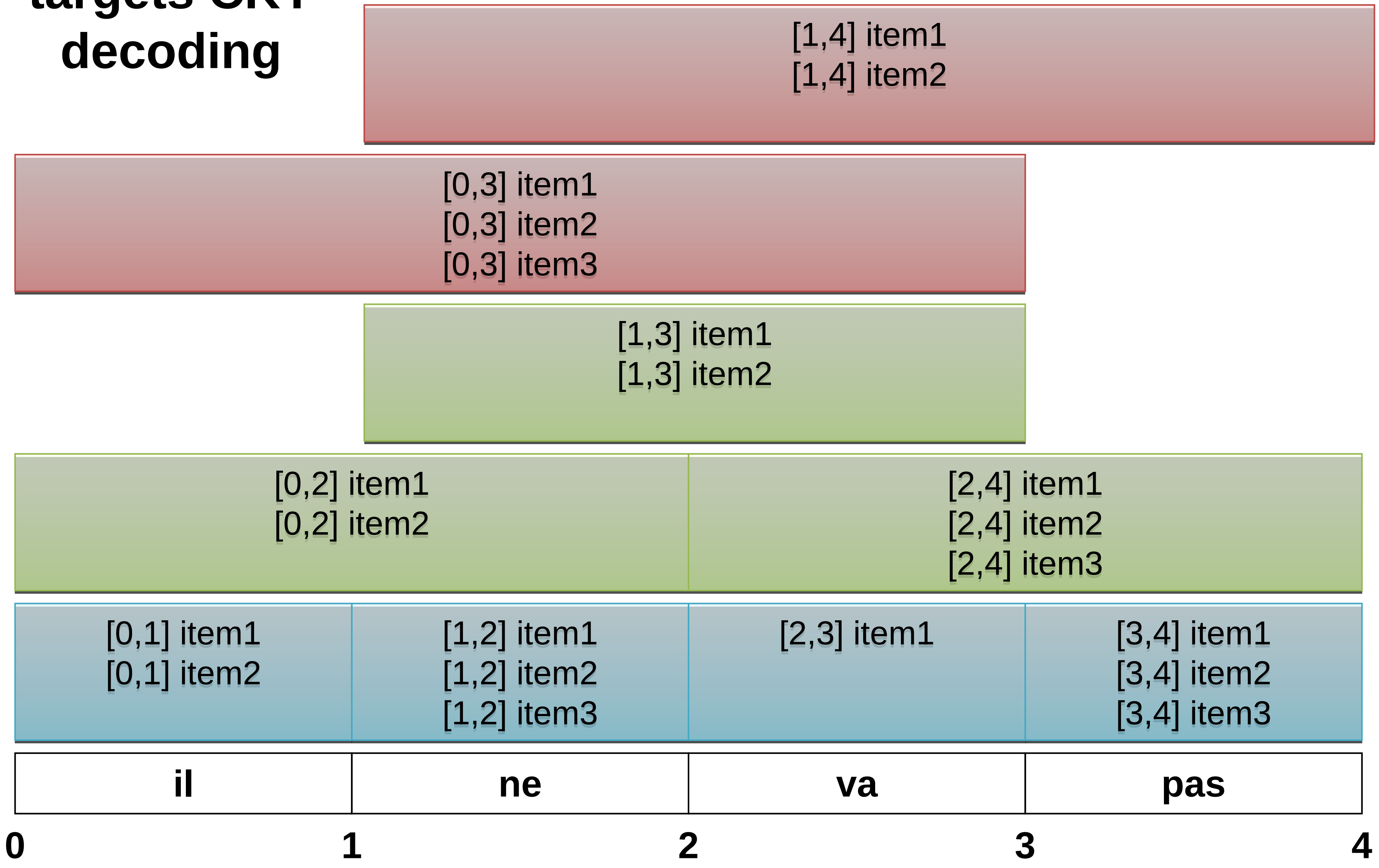
Mark Hopkins and Greg Langmead

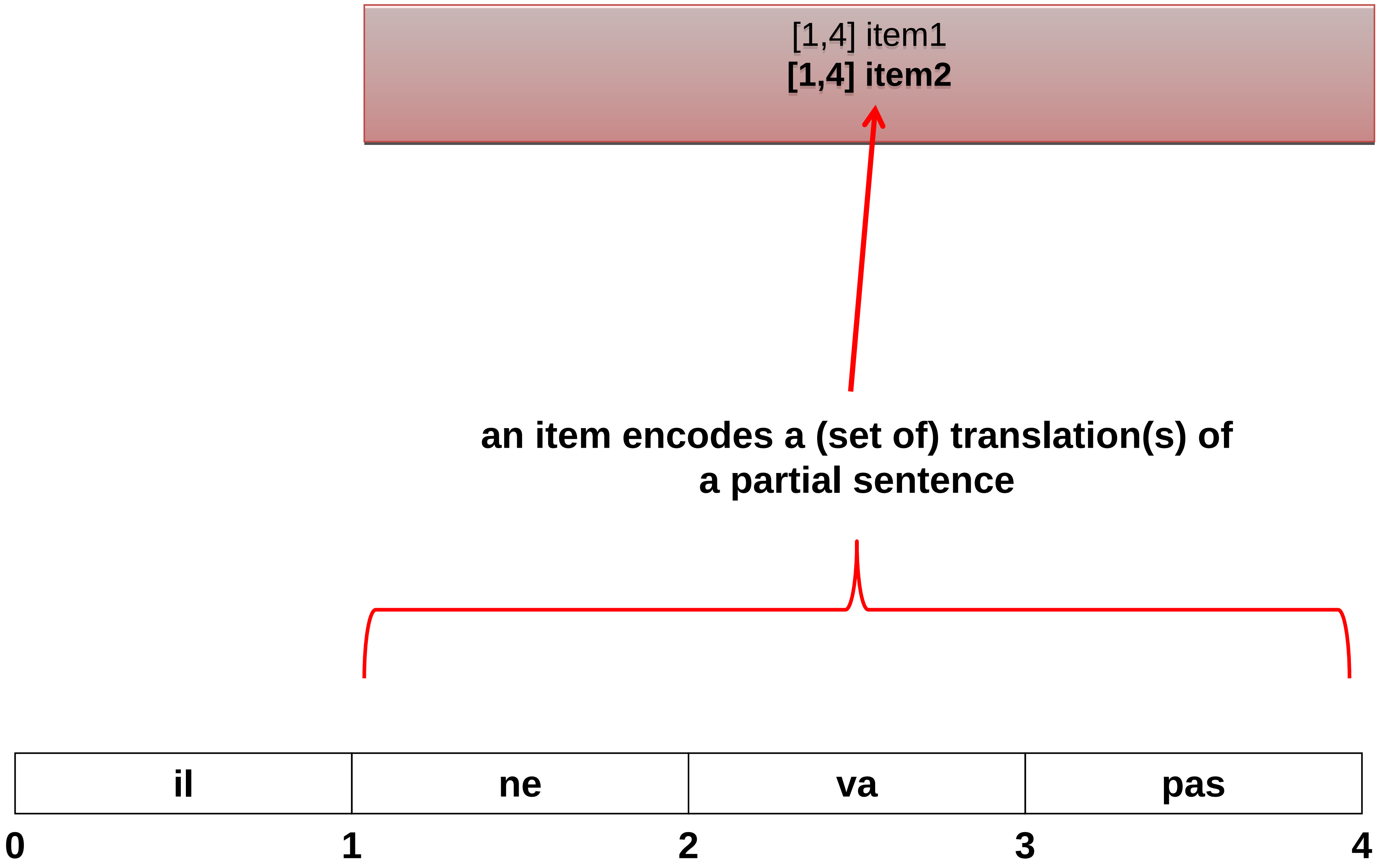
Language Weaver, Inc.

Motivation: Speed vs Accuracy

- Accuracy (e.g., BLEU) is very important
- But time is money
  - for the customer: throughput
  - for LW: cpu time on servers for SaaS solution
- Our customers expect 1000-3000 words per minute in one thread
  - and linear speedup with multiple threads
- That's 0.3-1.0 seconds per sentence
- Can syntax be viable at product speeds?

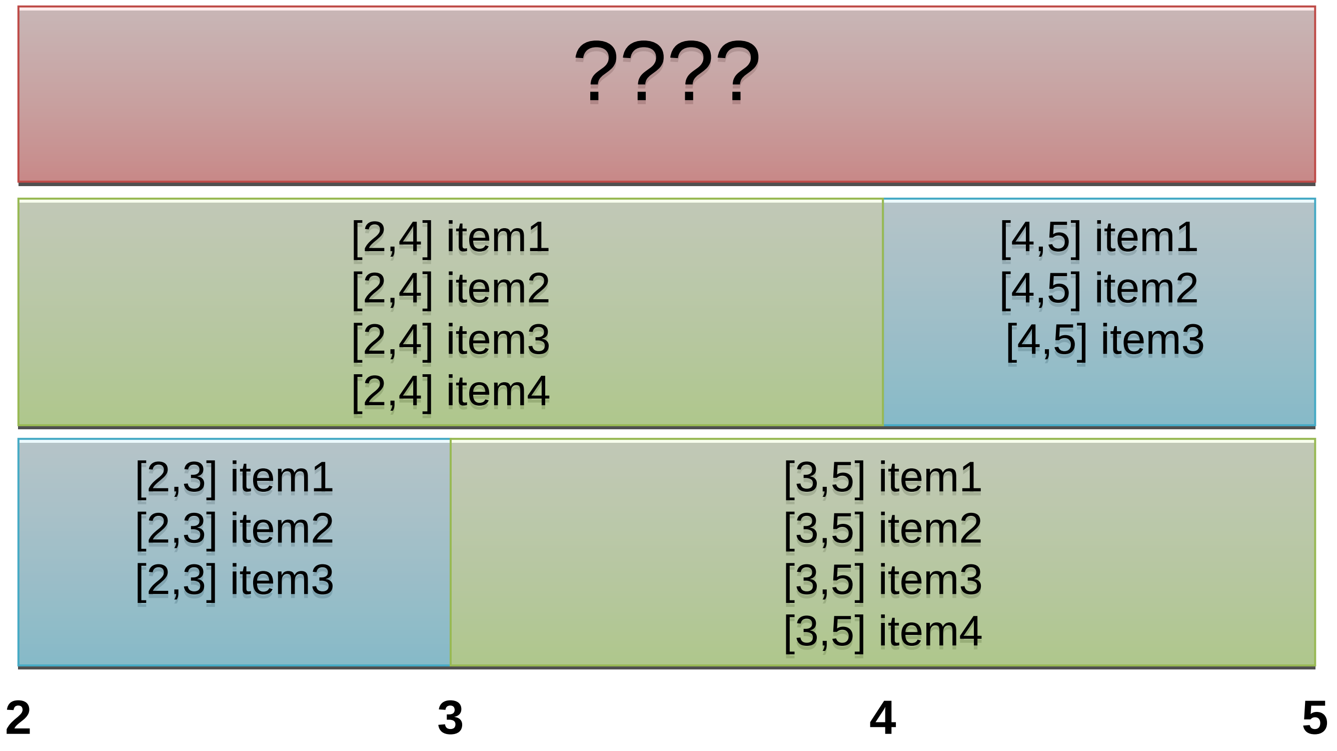
Cube pruning targets CKY decoding





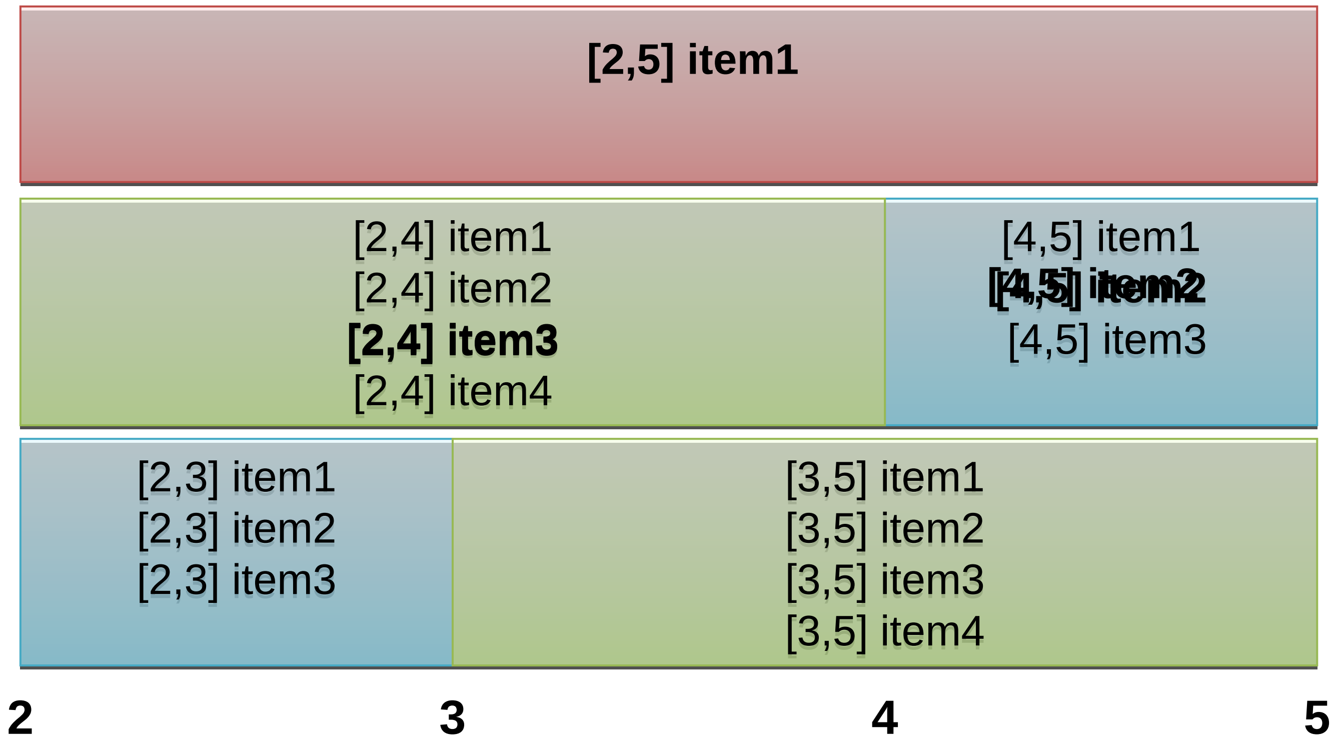
an item encodes a (set of) translation(s) of a partial sentence

How are items created?



LANGUAGE WEAVER

Items are created by combining items from complementary spans.



What is an item?

[2,4 , NP , the*car]

an item consists of three parts:

span   postcondition   carry

What is an item?

[2,4 , NP , the*car]

an item consists of three parts:

span   postcondition   carry

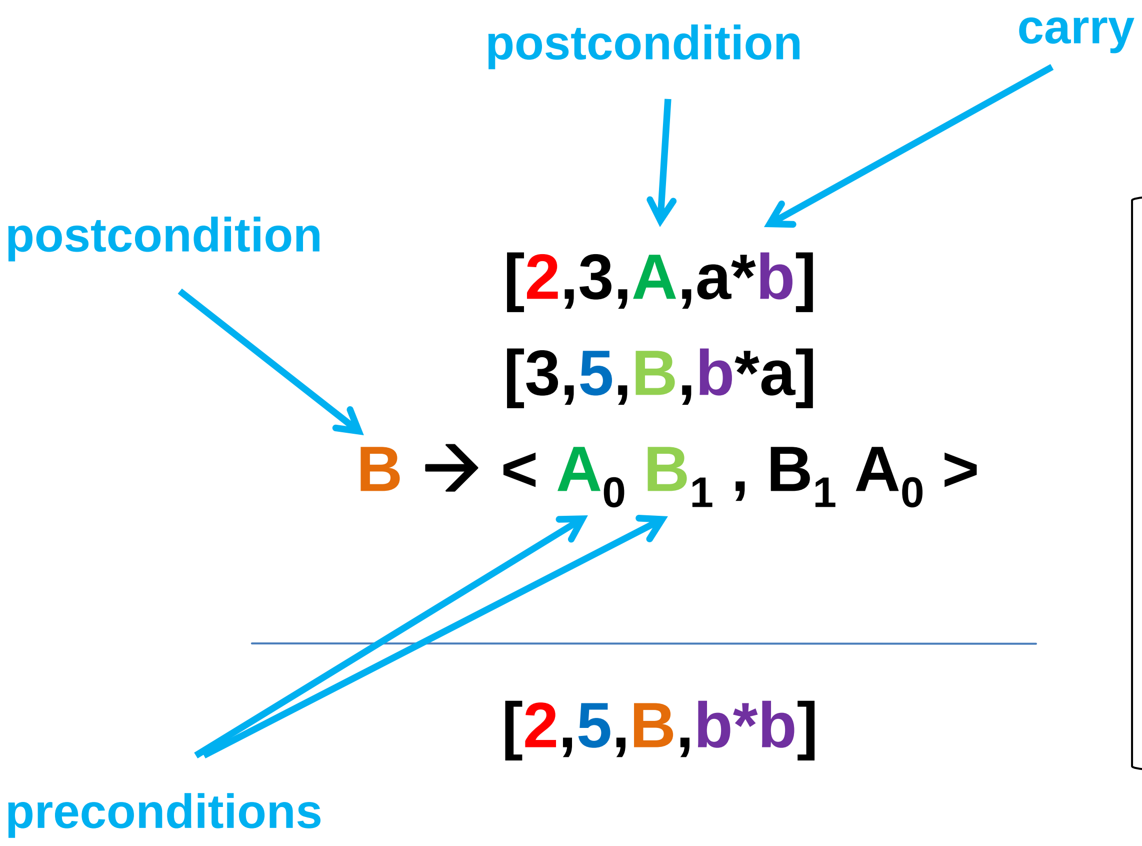
What is an item?

[2,4 , NP , the*car]

an item consists of three parts:

span postcondition carry

CKY Item Creation



$$\frac{[2,3,A,a*b] \qquad [3,5,B,b*a] \qquad B \rightarrow \langle A_0\ B_1 ,\ B_1\ A_0 \rangle}{[2,5,B,b*b]}$$

$$\left[\frac{\begin{array}{c}\text{cost(subitem1)}\\ \text{cost(subitem2)}\\ \text{cost(rule)}\\ \text{+ interaction(rule, subitems)}\end{array}}{\text{cost(new item)}}\right]$$

The Item Creation Problem

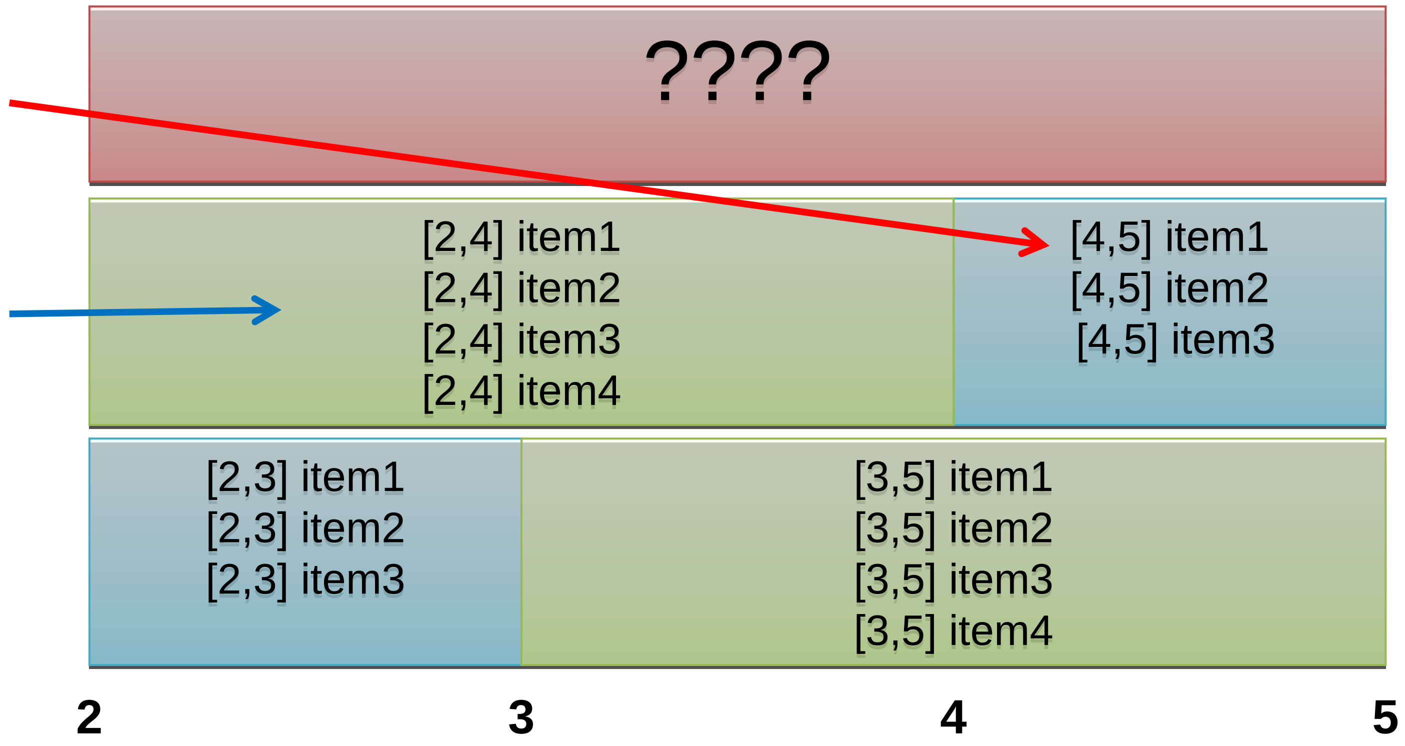
1000 1000 1000 000 1000

…then item creation can still take a very long time

even if we just store 1000 items here

…and 1000 items here

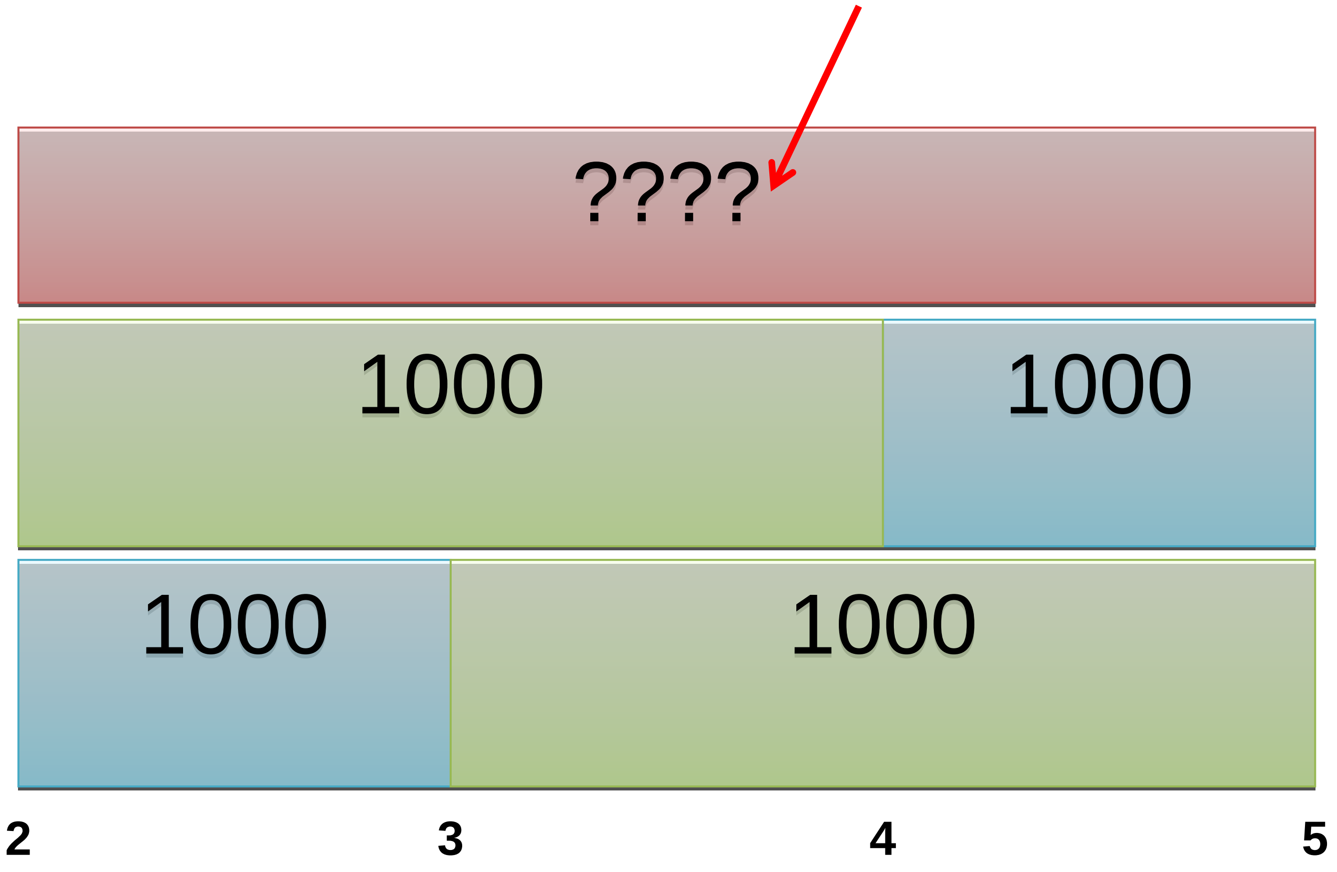
…and we have only 1000s of grammar rules



The Item Creation Problem

is there a better way to enumerate the 1000 items of lowest cost for this span without going through the millions of candidate items and taking the best 1000?

this is the problem that cube pruning addresses



A demonstration of incremental CKY item creation for span [2,5]

We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1 , B_1\ A_0 >$

[2,5,B,b*b]

So far we have:

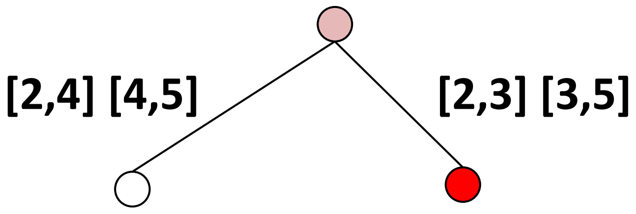
[?,?,?,?]

[?,?,?,?]

$? \rightarrow < ?_0\ ?_1 , ?\ ? >$

[?,?,?,?]

LANGUAGE WEAVER



We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1\ ,\ B_1\ A_0 >$

[2,5,B,b*b]

So far we have:

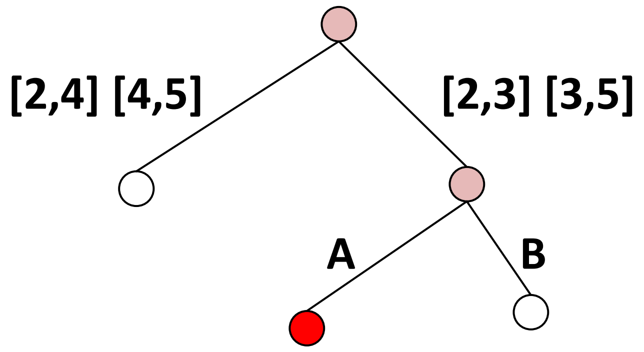
[2,3,?,?]

[3,5,?,?]

$? \rightarrow < ?_0\ ?_1\ ,\ ?\ ?\ >$

[2,5,?,?]

LANGUAGE WEAVER



We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1\ ,\ B_1\ A_0 >$

[2,5,B,b*b]

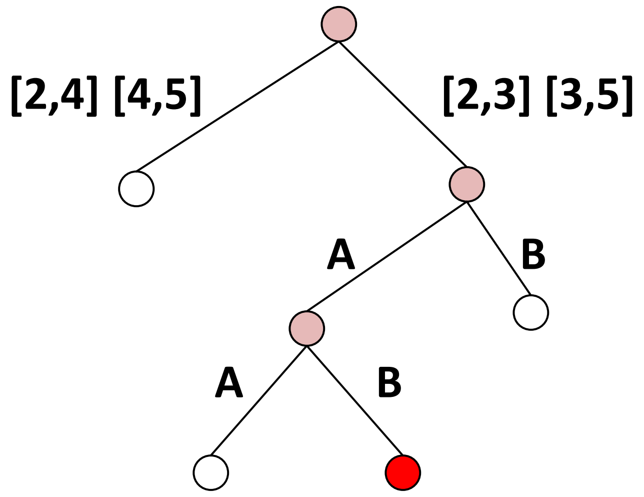
So far we have:

[2,3,A,?]

[3,5,?,?]

$? \rightarrow < A_0\ ?_1\ ,\ ?\ ?>$

[2,5,?,?]



We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1\ ,\ B_1\ A_0 >$

[2,5,B,b*b]

So far we have:

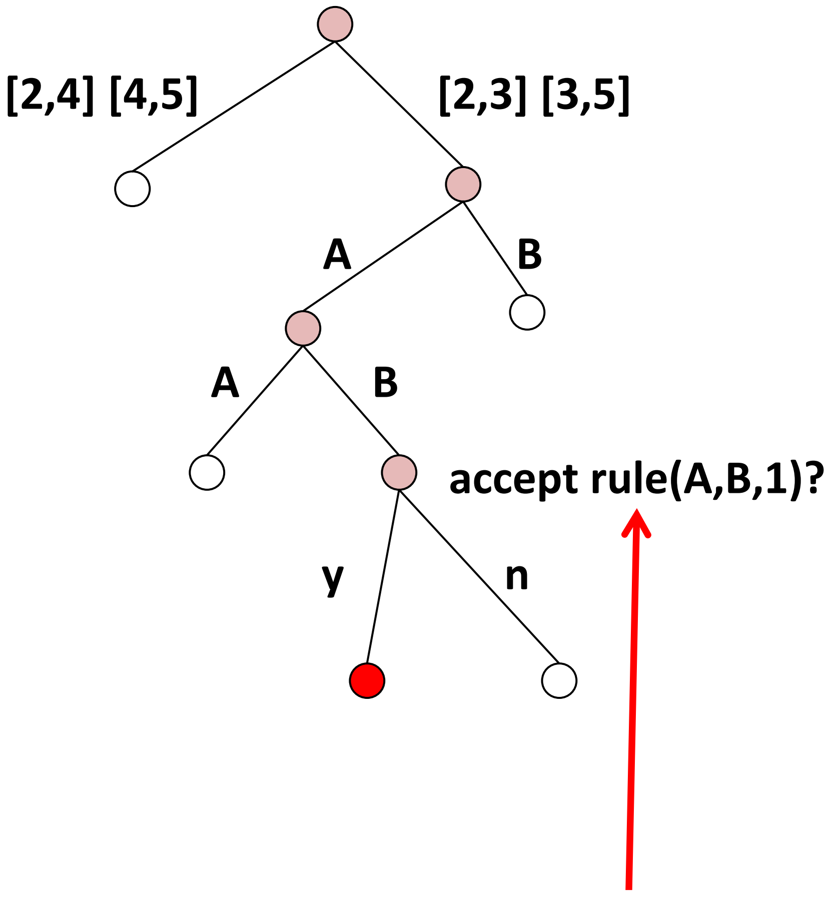
[2,3,A,?]

[3,5,B,?]

$? \rightarrow < A_0\ B_1\ ,\ ?\ ?\ >$

[2,5,?,?]

LANGUAGE WEAVER



rule(A,B,k) is the kth lowest cost rule whose preconditions are <A,B>

We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1\ ,\ B_1\ A_0 >$

[2,5,B,b*b]

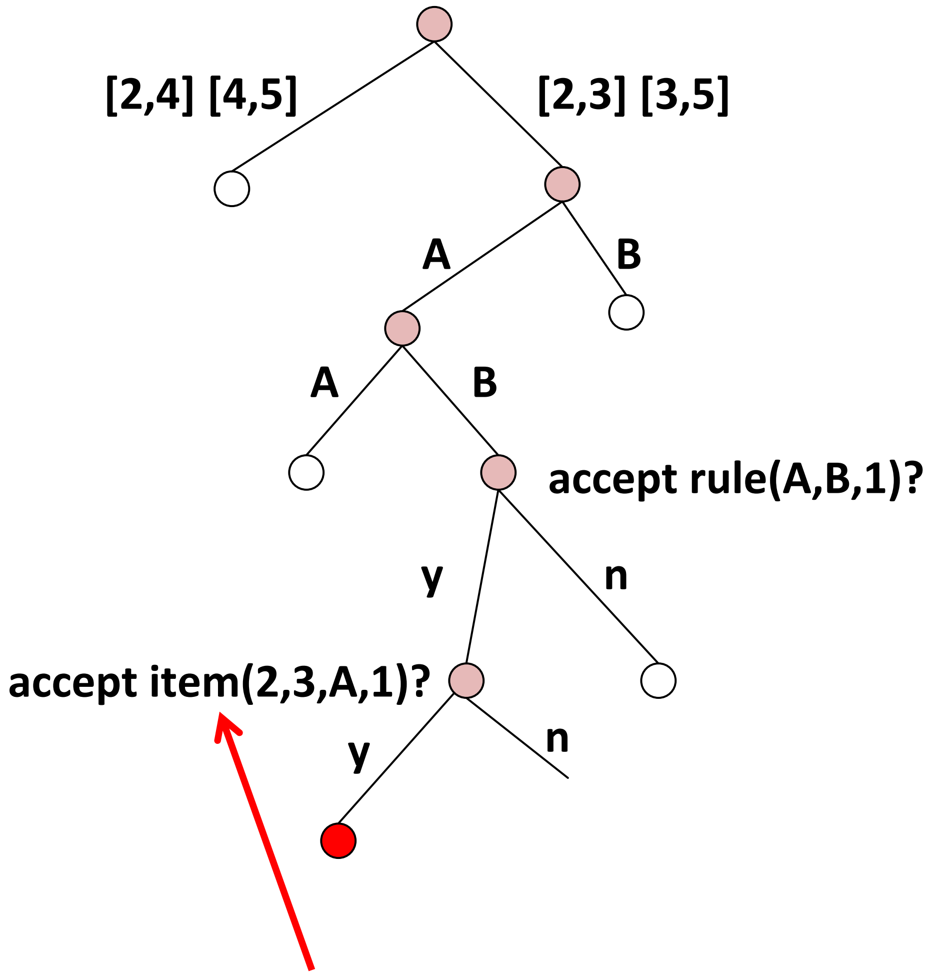
So far we have:

[2,3,A,?]

[3,5,B,?]

$B \rightarrow < A_0\ B_1\ ,\ B_1\ A_0 >$

[2,5,B,?]



item(2,3,A,k) is the kth lowest cost item of span [2,3] whose postcondition is A

We want:

[2,3,A,a*b]

[3,5,B,b*a]

B → < A_0 B_1 , B_1 A_0 >

[2,5,B,b*b]

So far we have:

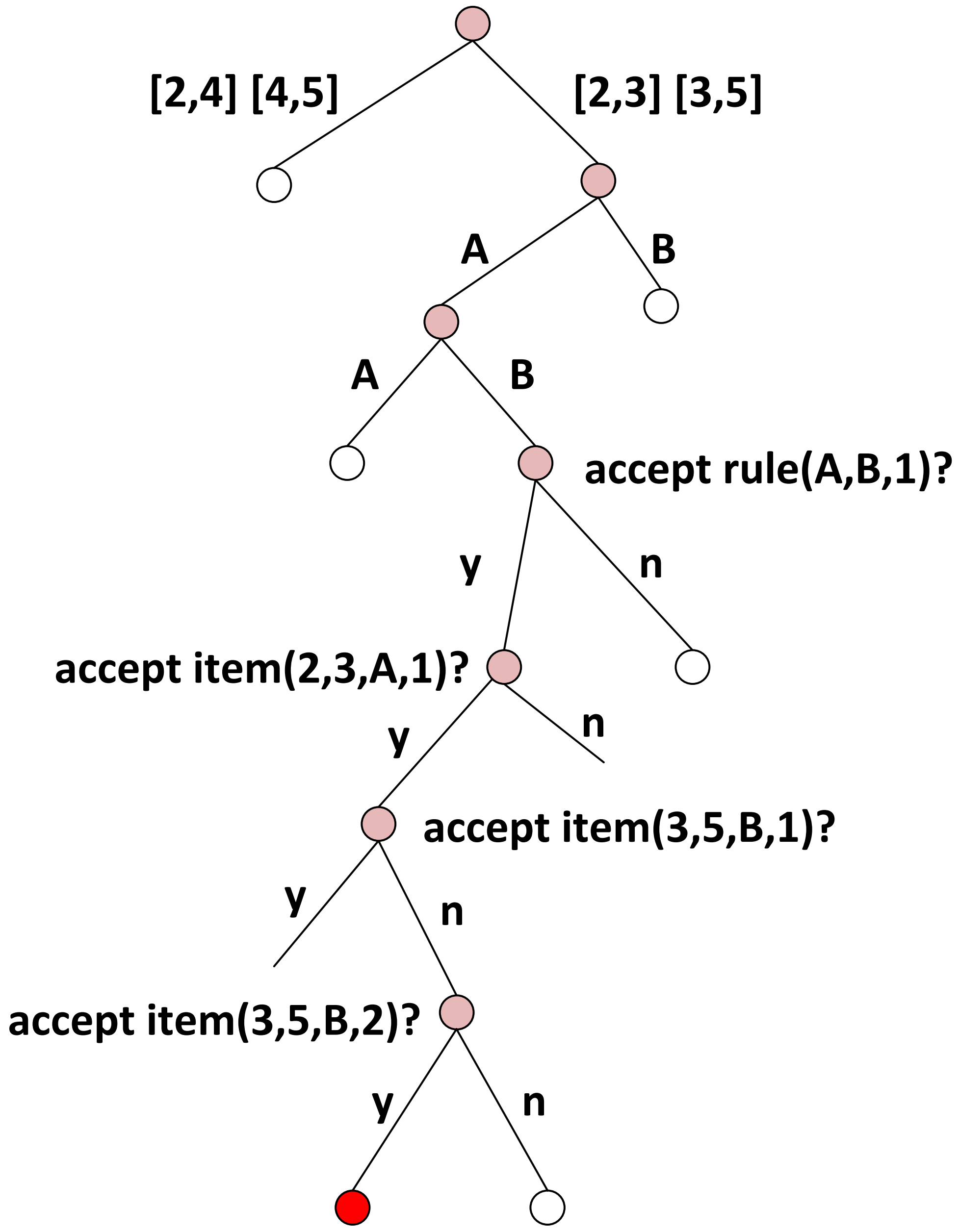
[2,3,A,a*b]

[3,5,B,?]

B → < A_0 B_1 , B_1 A_0 >

[2,5,B,?*b]

LANGUAGE WEAVER



We want:

[2,3,A,a*b]

[3,5,B,b*a]

$B \rightarrow < A_0\ B_1 , B_1\ A_0 >$

[2,5,B,b*b]

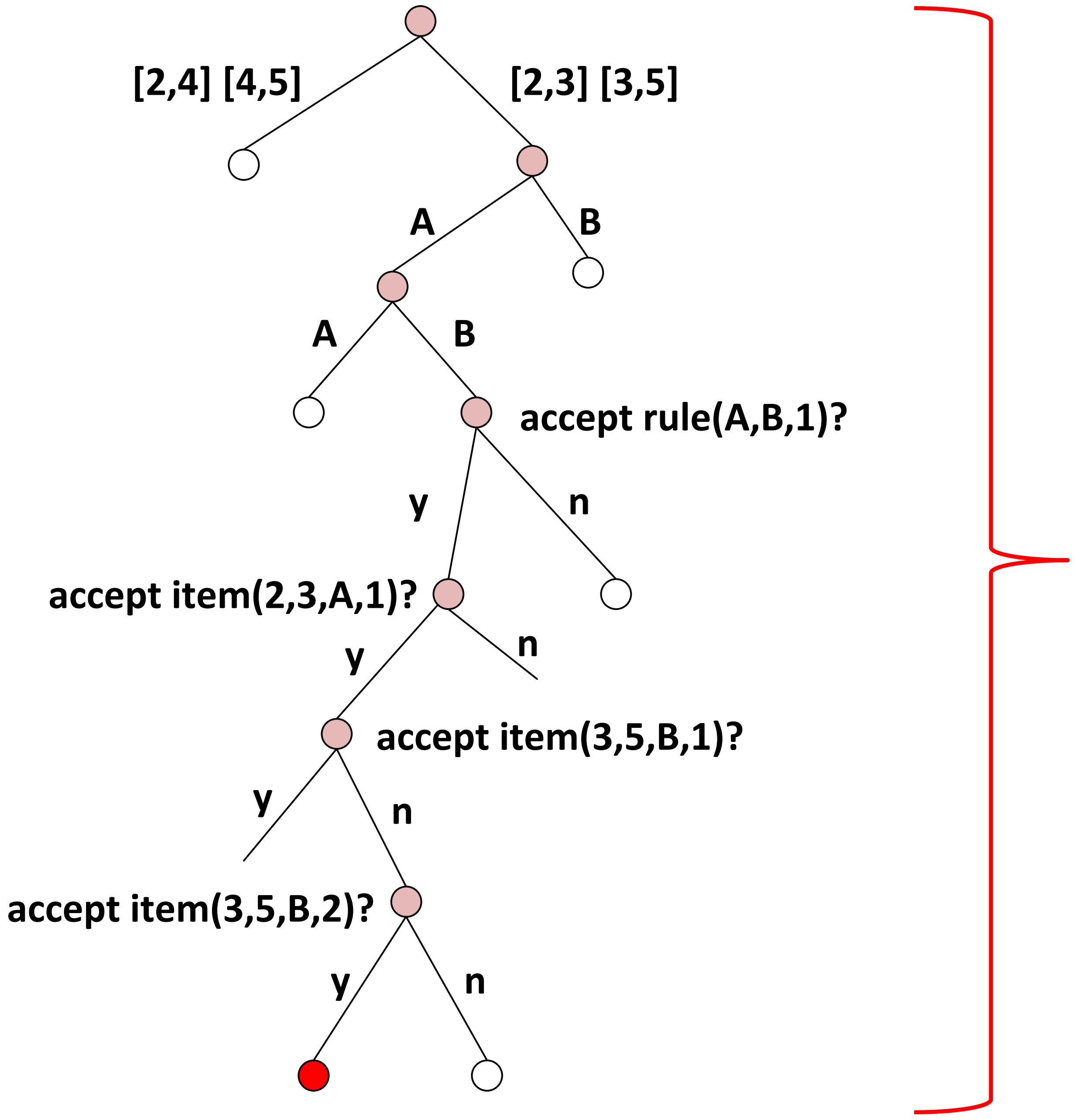
So far we have:

[2,3,A,a*b]

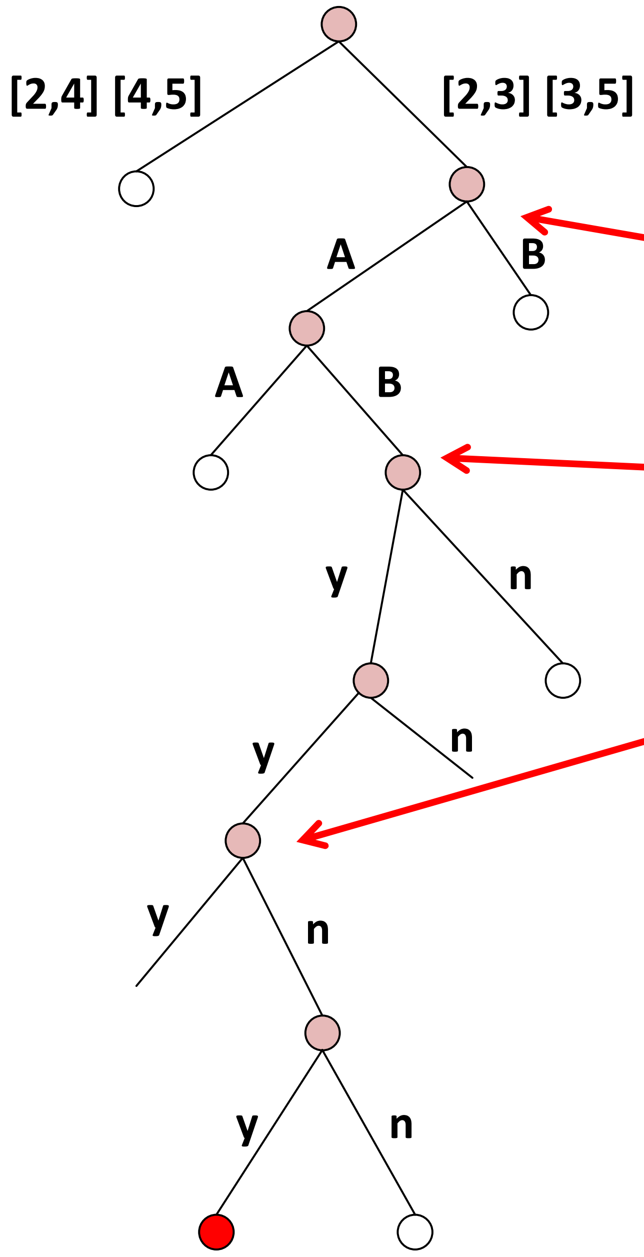
[3,5,B,b*a]

$B \rightarrow < A_0\ B_1 , B_1\ A_0 >$

[2,5,B,b*b]



The Item Creation Problem, rephrased: find the n lowest-cost goal nodes of this search space



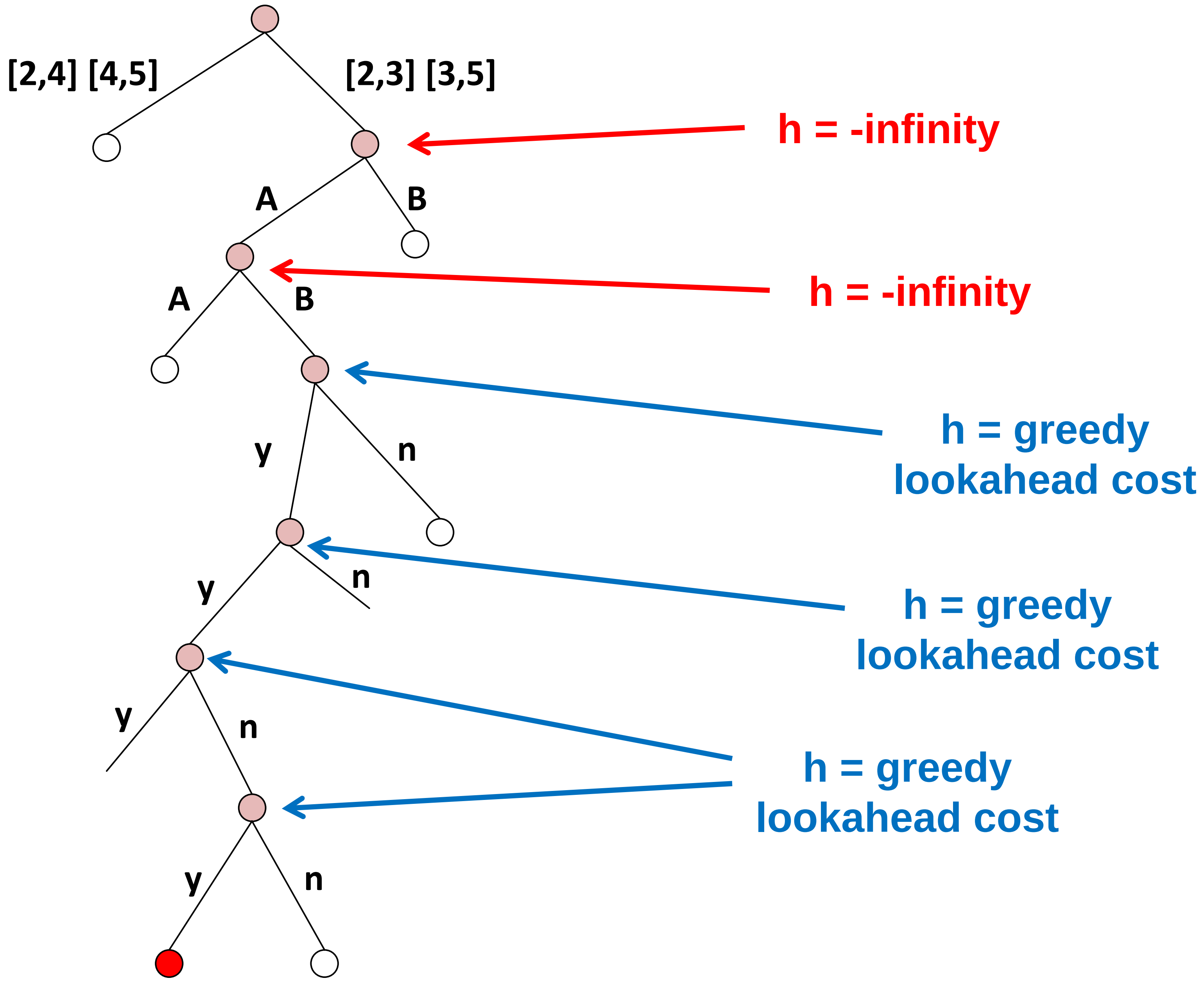
so if we can come up with lower-bounds on the best-cost reachable goal node from here…

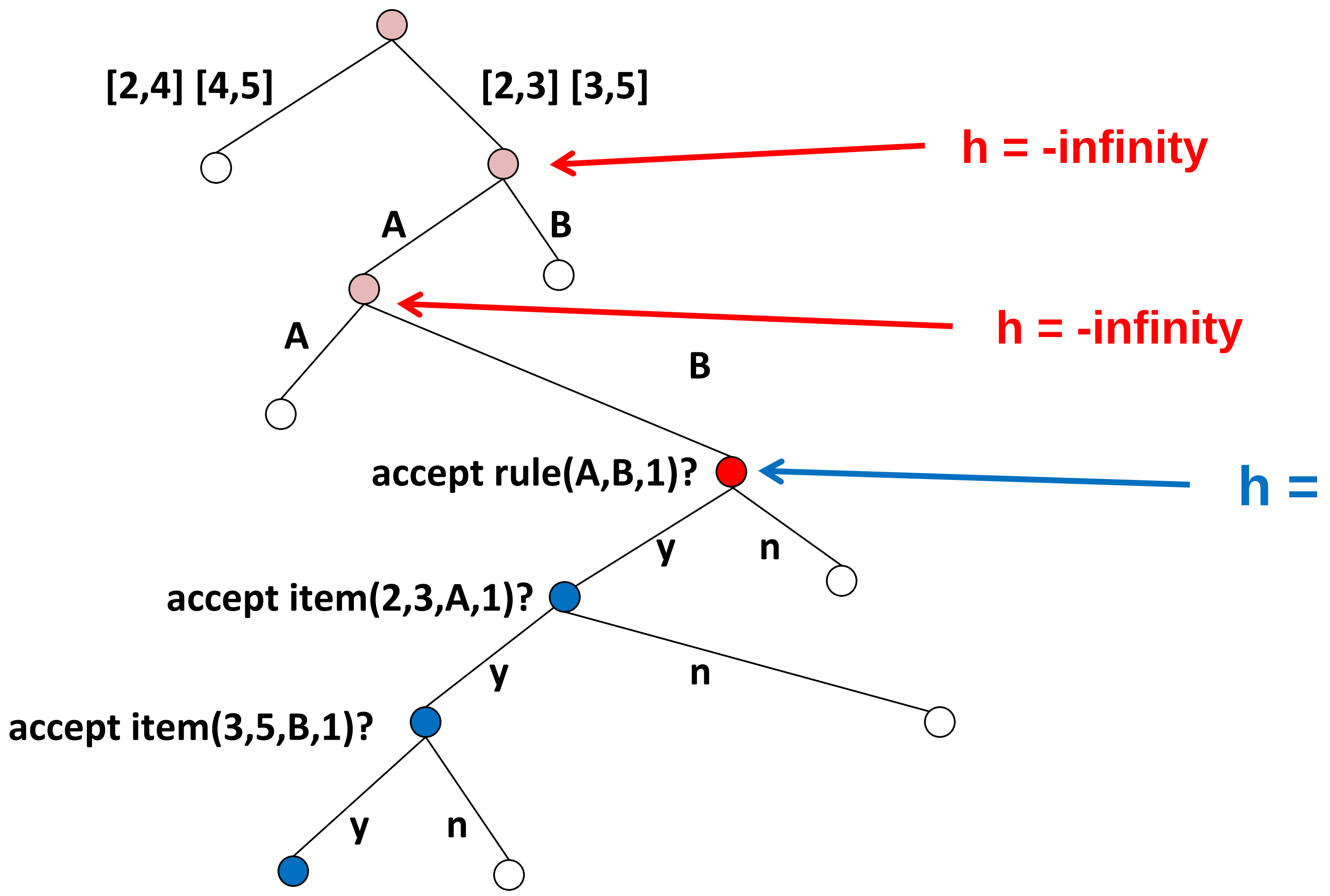
…and here

…and here

…then we can just run A* on this search space to find the n goal nodes of lowest cost (without searching the entire space)

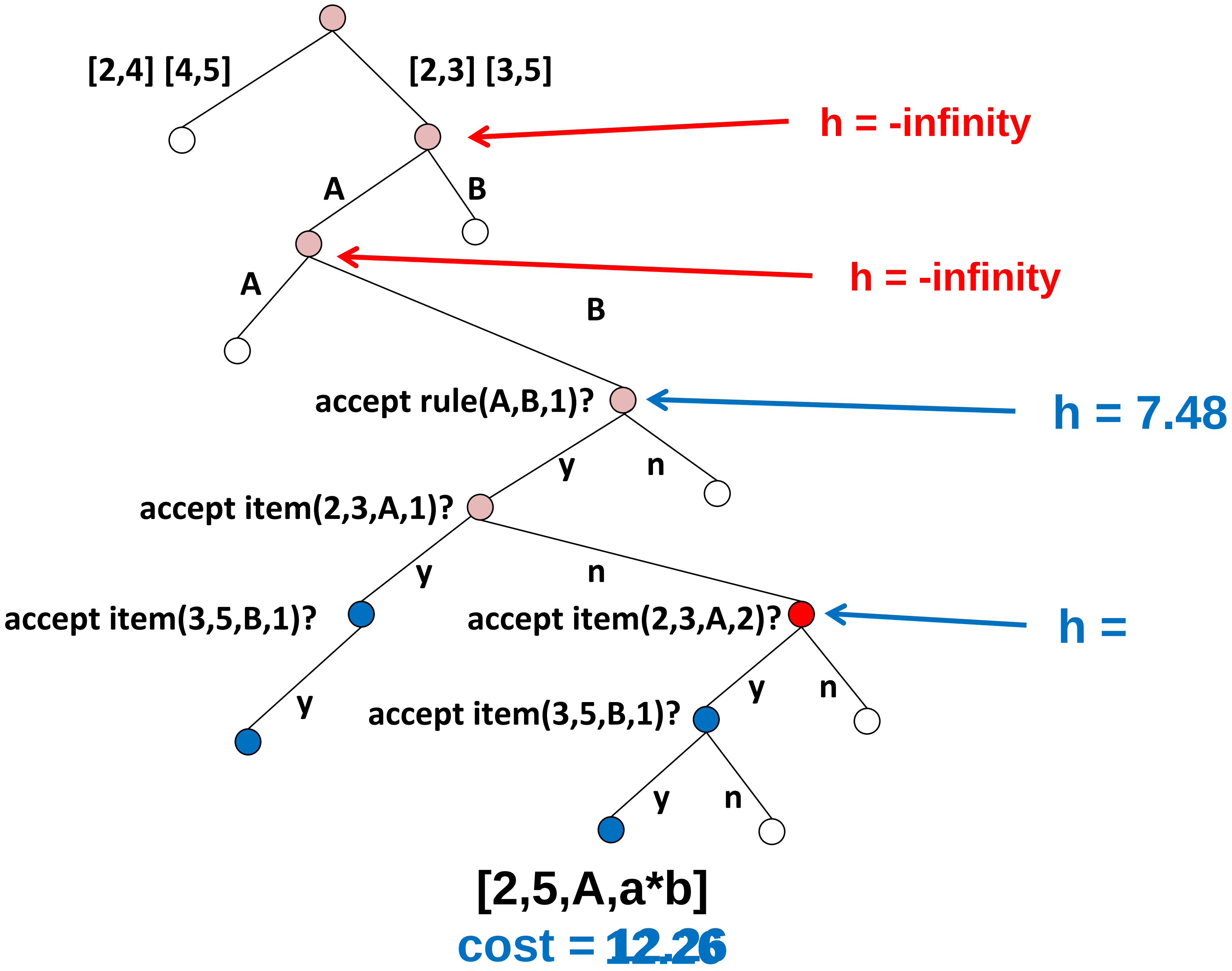
LANGUAGE WEAVER

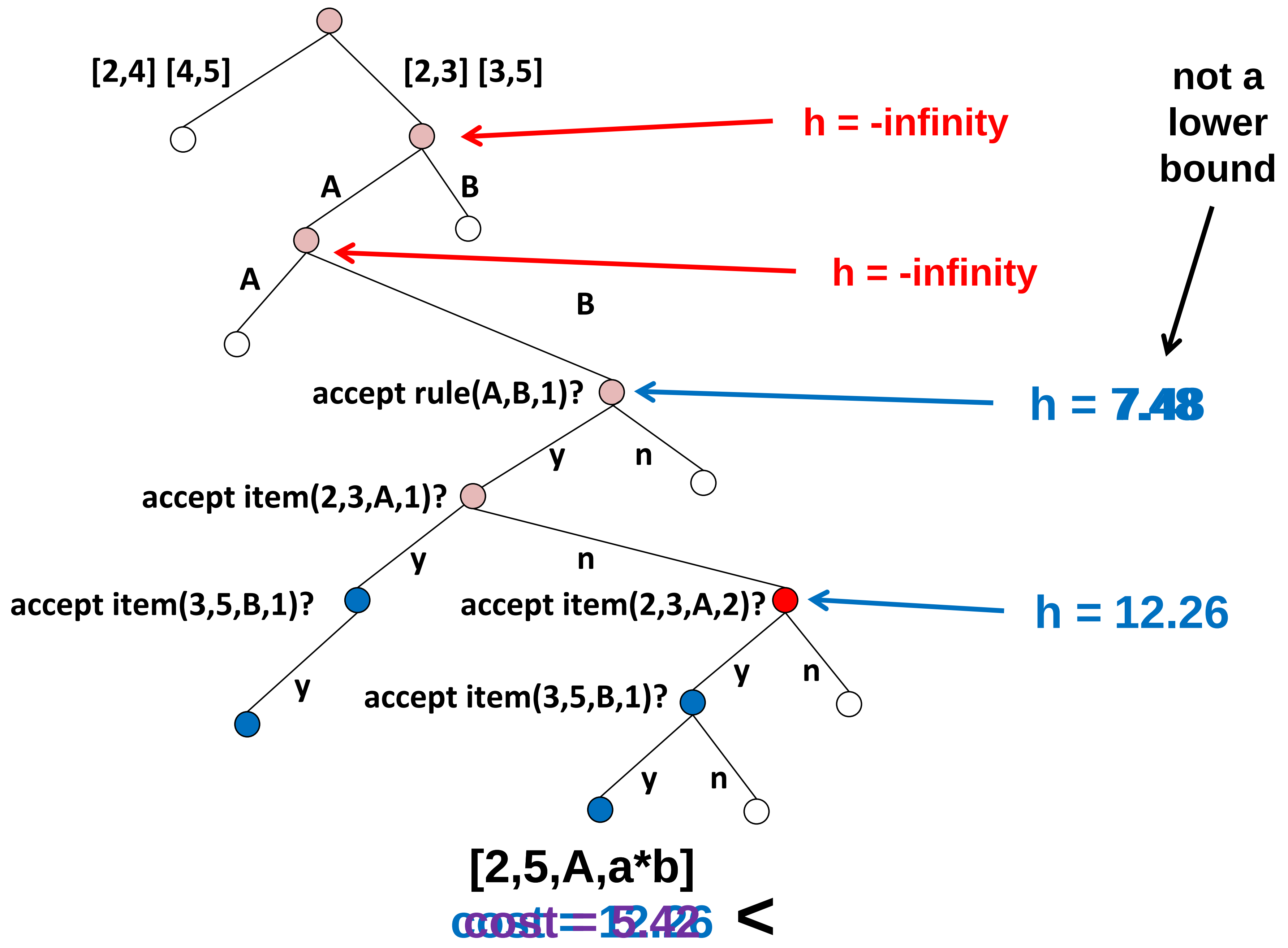

[2,4] [4,5]
[2,3] [3,5]
h = -infinity
A
B
A
B
h = -infinity
y
n
h = greedy lookahead cost
y
n
h = greedy lookahead cost
y
n
h = greedy lookahead cost
y
n

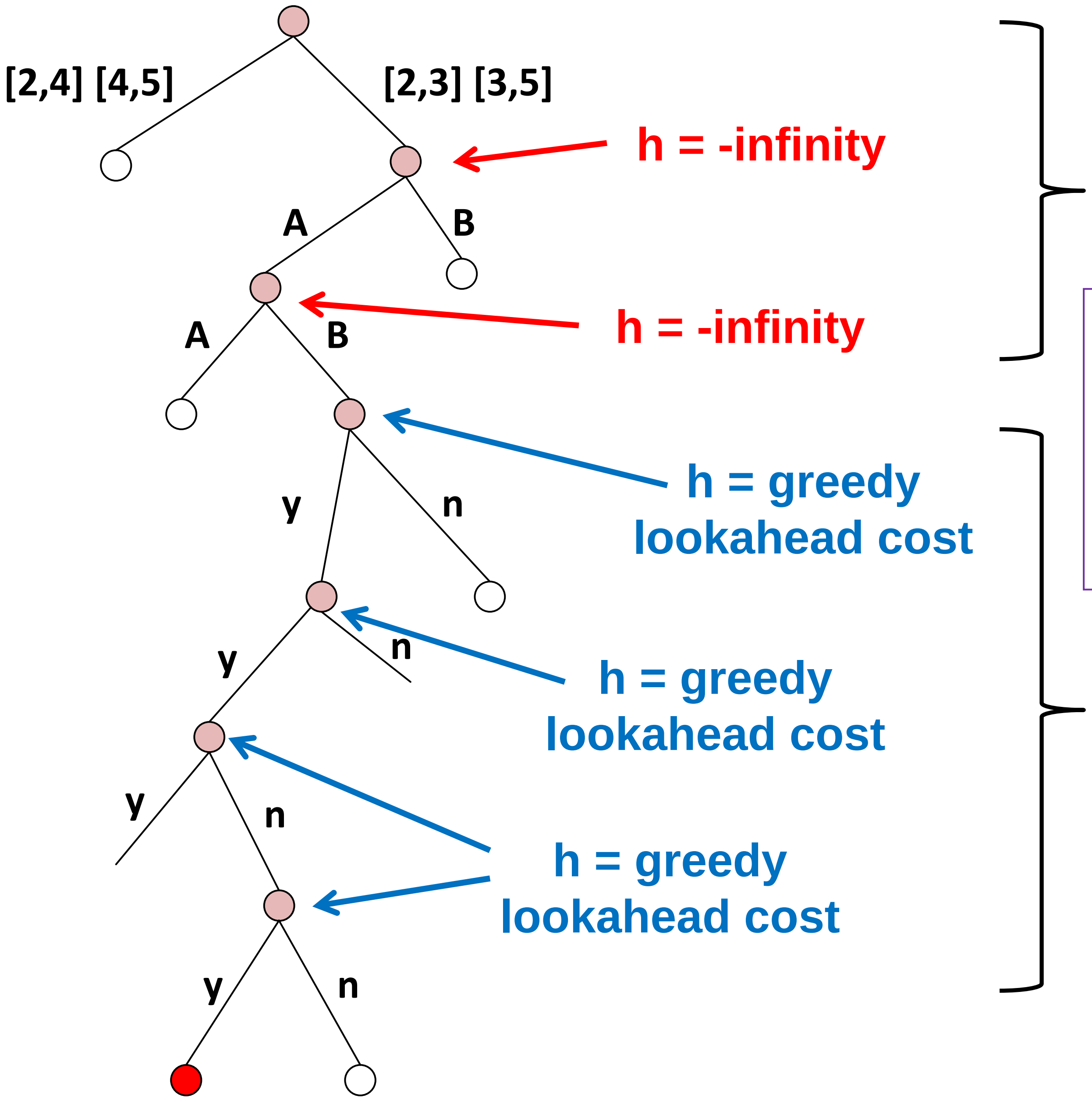


[2,5,B,b*b]

cost = 7.48


[2,4] [4,5]
[2,3] [3,5]
h = -infinity
A
B
A
h = -infinity
B
accept rule(A,B,1)?
h = 7.48
y
n
accept item(2,3,A,1)?
y
n
accept item(3,5,B,1)?
accept item(2,3,A,2)?
h =
y
y
n
accept item(3,5,B,1)?
y
n
[2,5,A,a*b]
cost = 12.26


[2,4] [4,5]
[2,3] [3,5]
h = -infinity
not a lower bound
A
B
A
h = -infinity
B
accept rule(A,B,1)?
h = 7.48
y
n
accept item(2,3,A,1)?
y
n
accept item(3,5,B,1)?
accept item(2,3,A,2)?
h = 12.26
y
y
n
accept item(3,5,B,1)?
y
n
[2,5,A,a*b]
cost = 12.26
cost = 5.42
<



therefore A* will not find the n best solutions, it will only find n good solutions

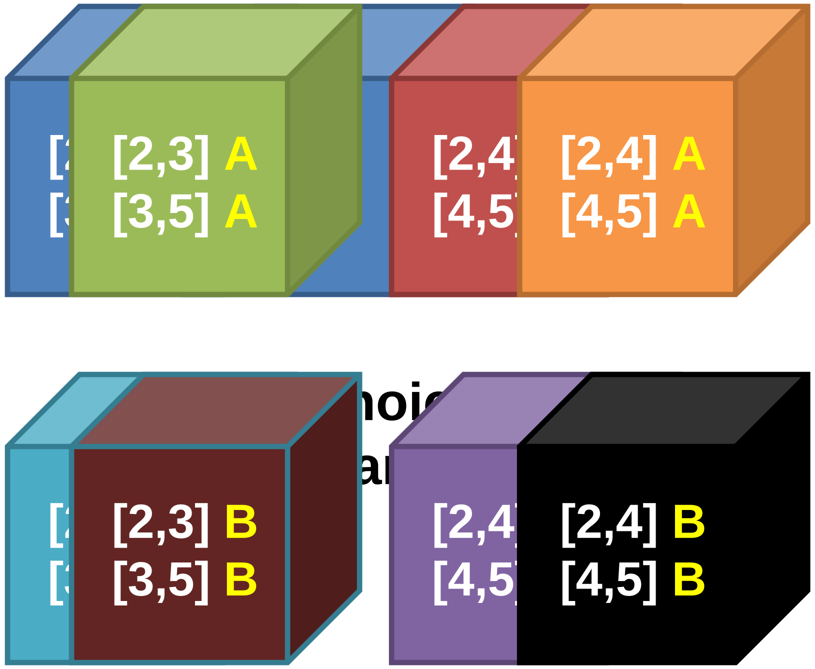
LANGUAGE WEAVER

A*
search

vs.

Cube
pruning

Cube pruning begins by forming cubes



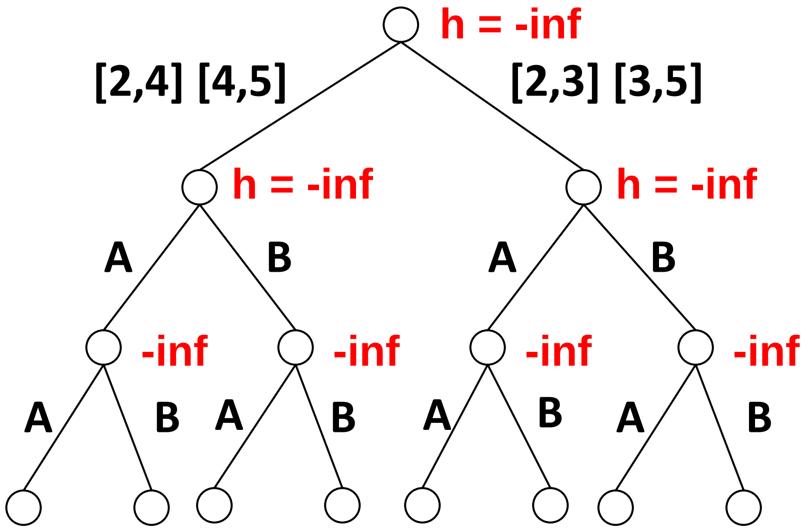
and preconditions

A* search

Cube pruning

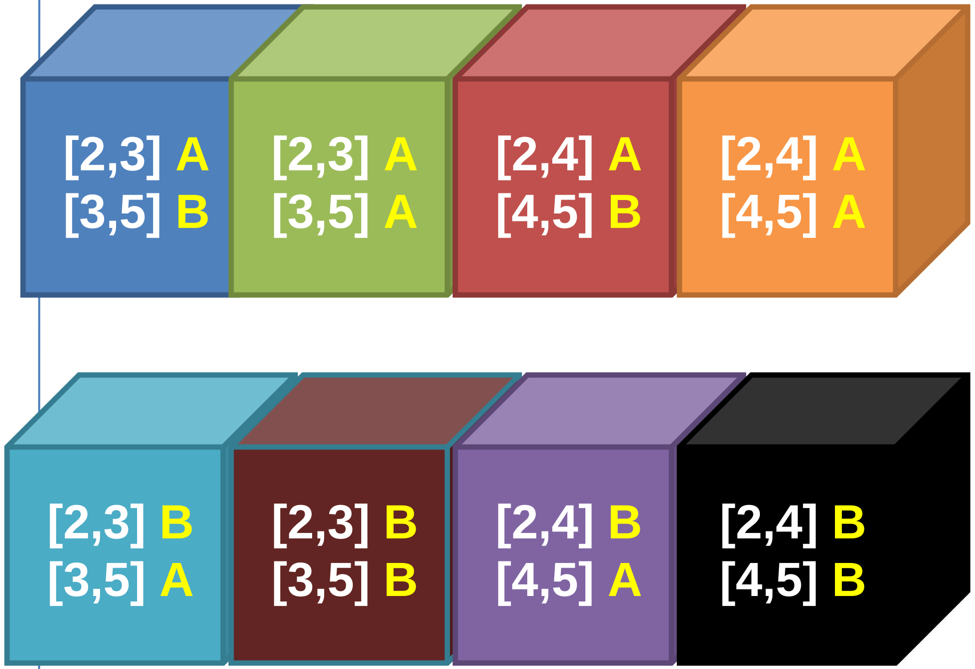
LANGUAGE WEAVER

A* search visits nodes in increasing order of heuristic value



therefore, it will begin by visiting all nodes with –inf heuristics

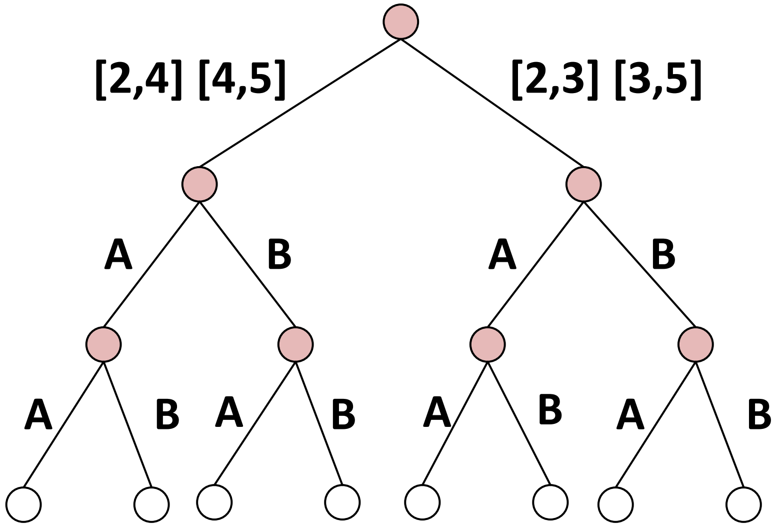
Cube pruning begins by forming cubes



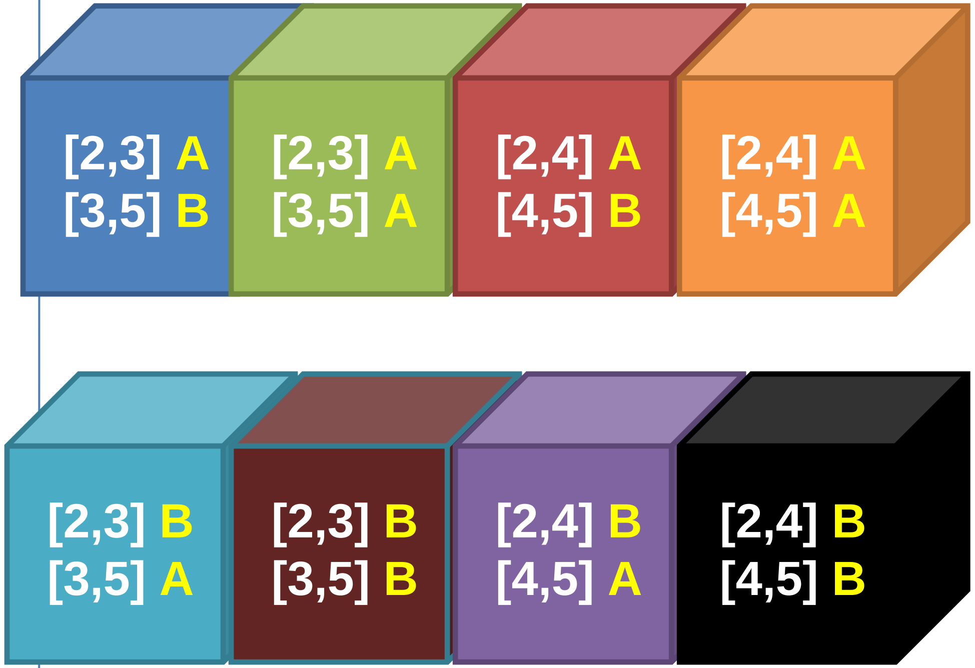
A* search

Cube pruning

A* search visits nodes in order of increasing heuristic value



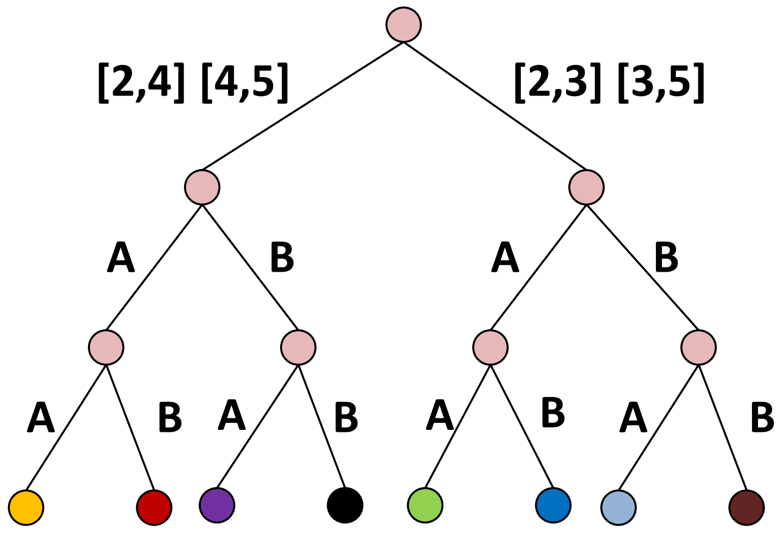
Cube pruning begins by forming cubes



A* search

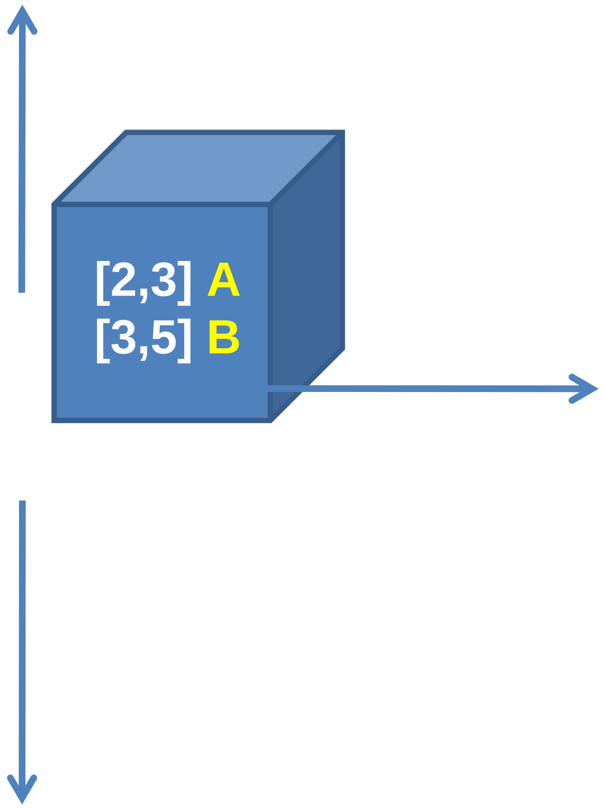
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

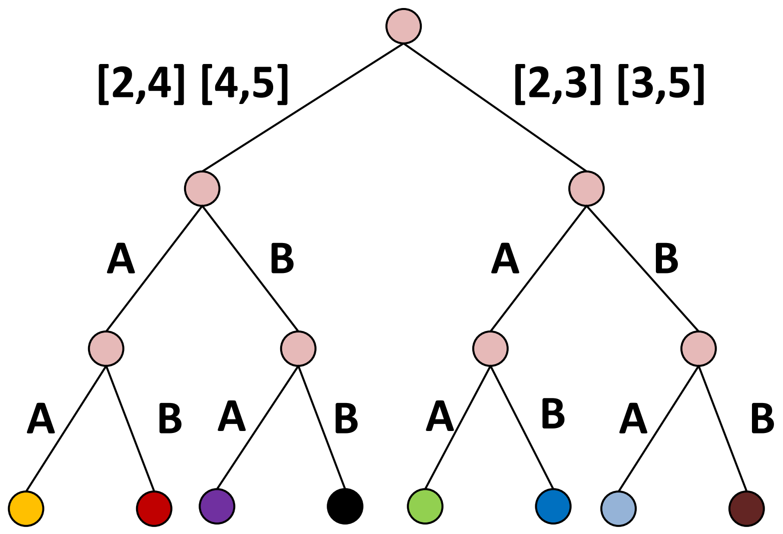
What is a cube?



a cube is a set of three axes

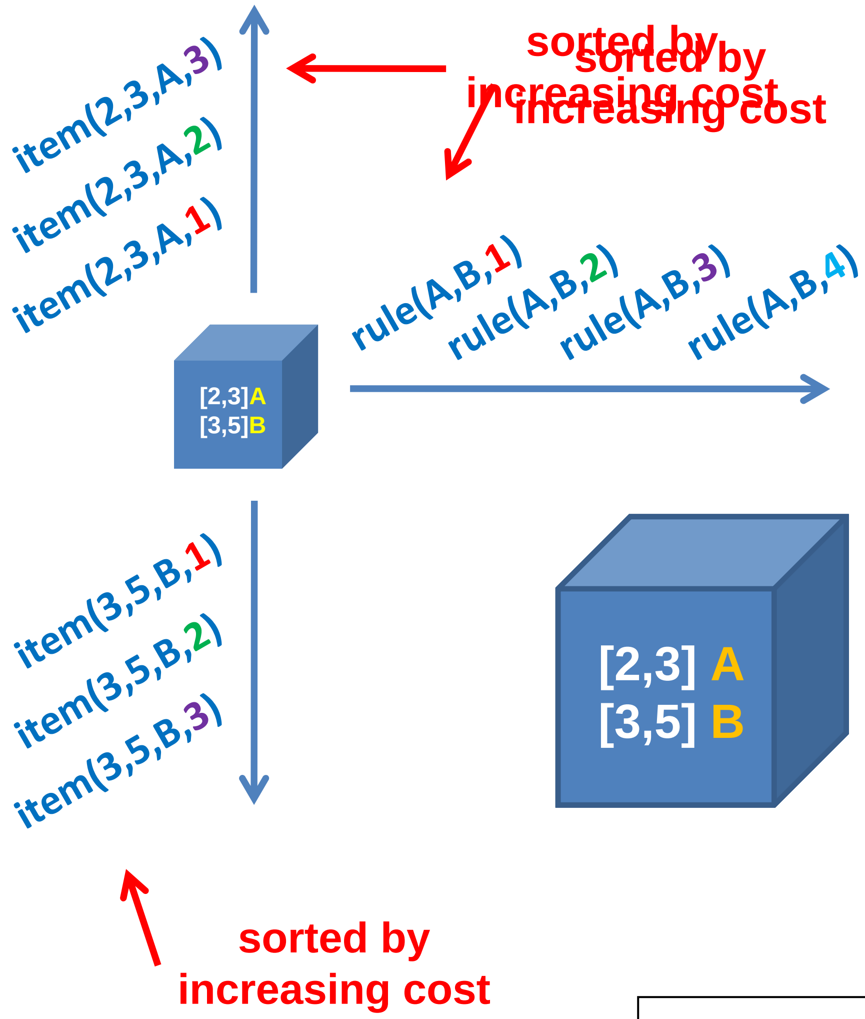
Cube pruning

A* search visits nodes in order of increasing heuristic value



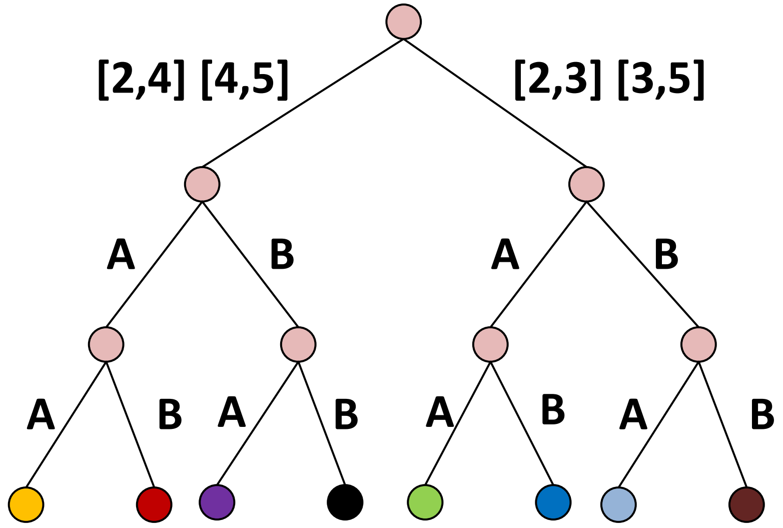
A* search

What is a cube?



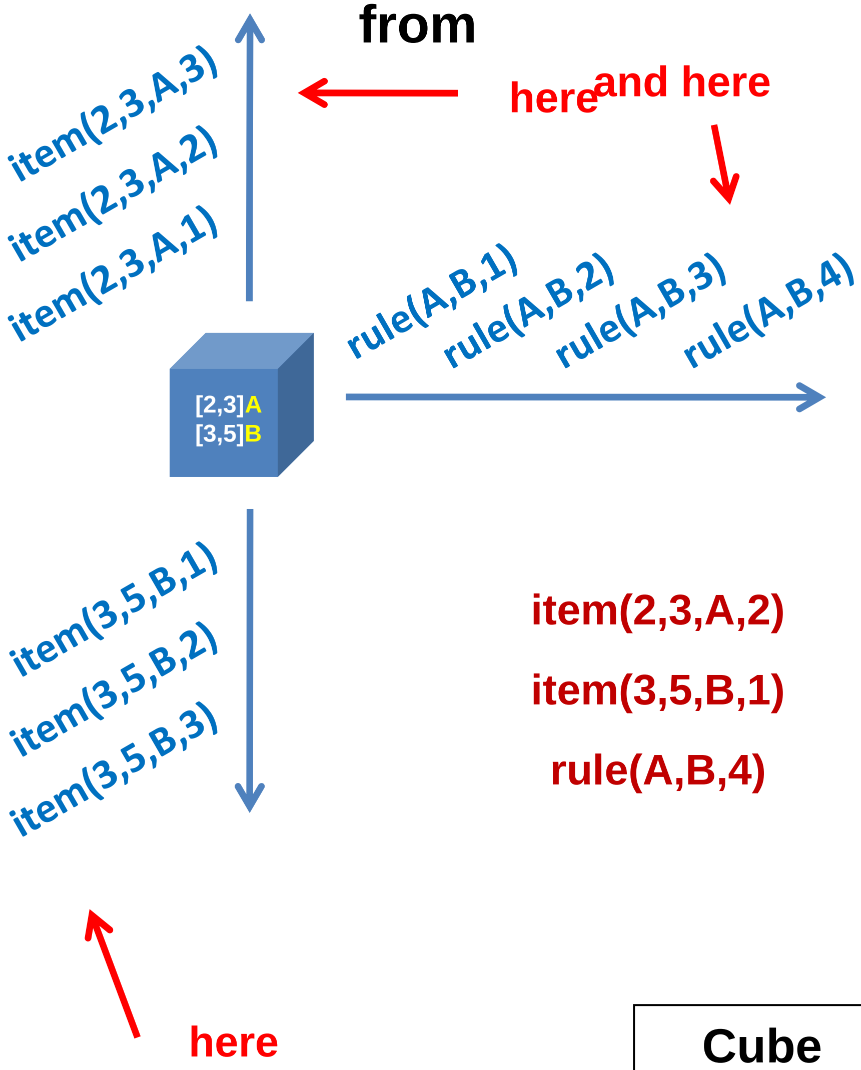
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

Thus each choice of object from



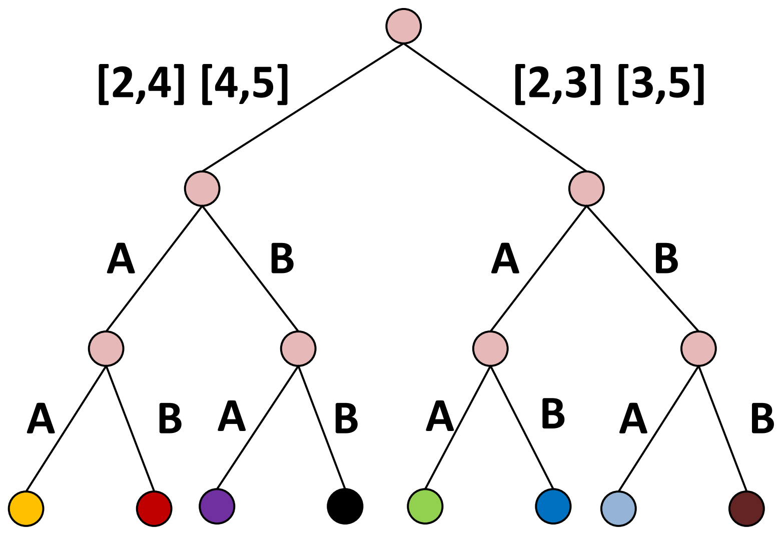
item(2,3,A,2)

item(3,5,B,1)

rule(A,B,4)

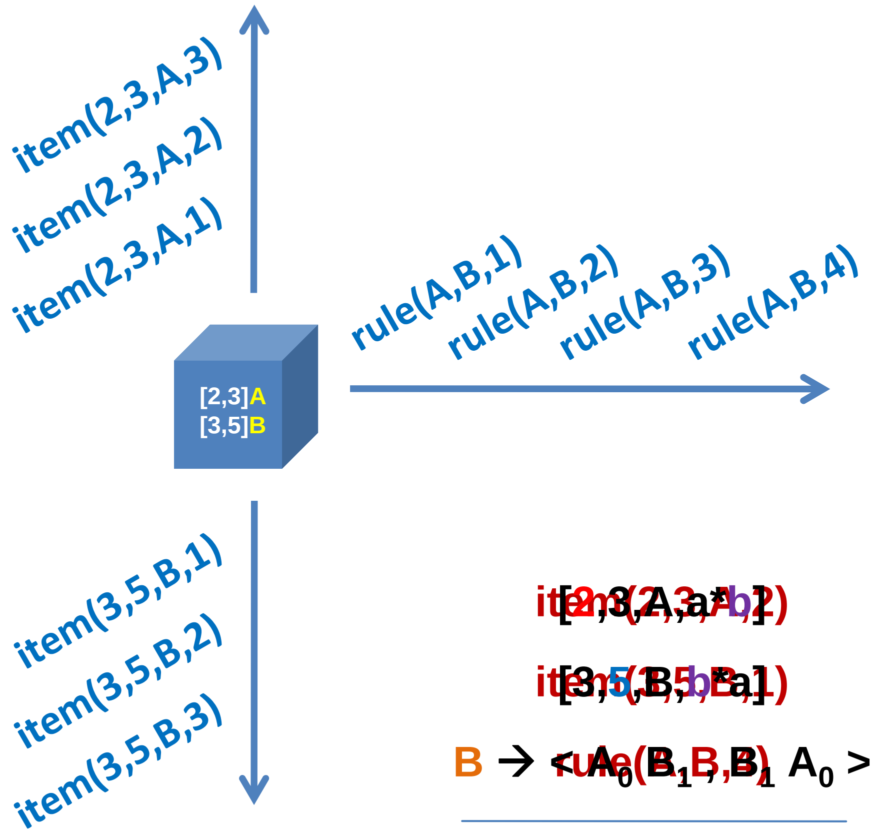
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

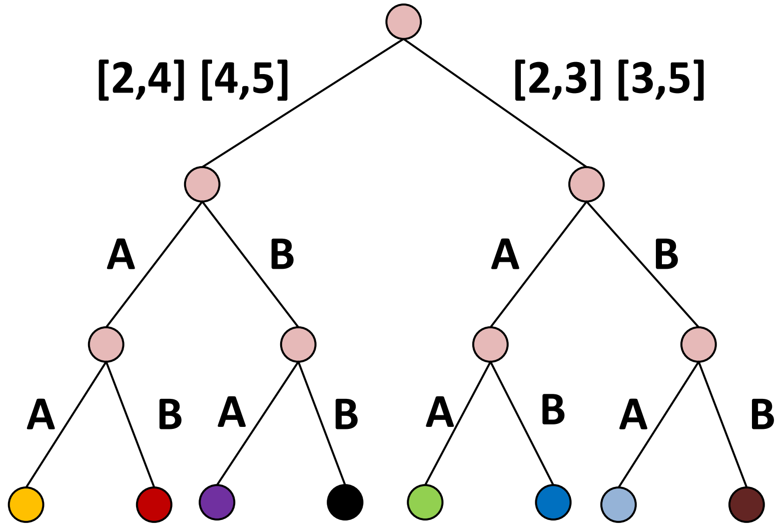
…creates a new item for [2,5]



[2,5,B,b*b]

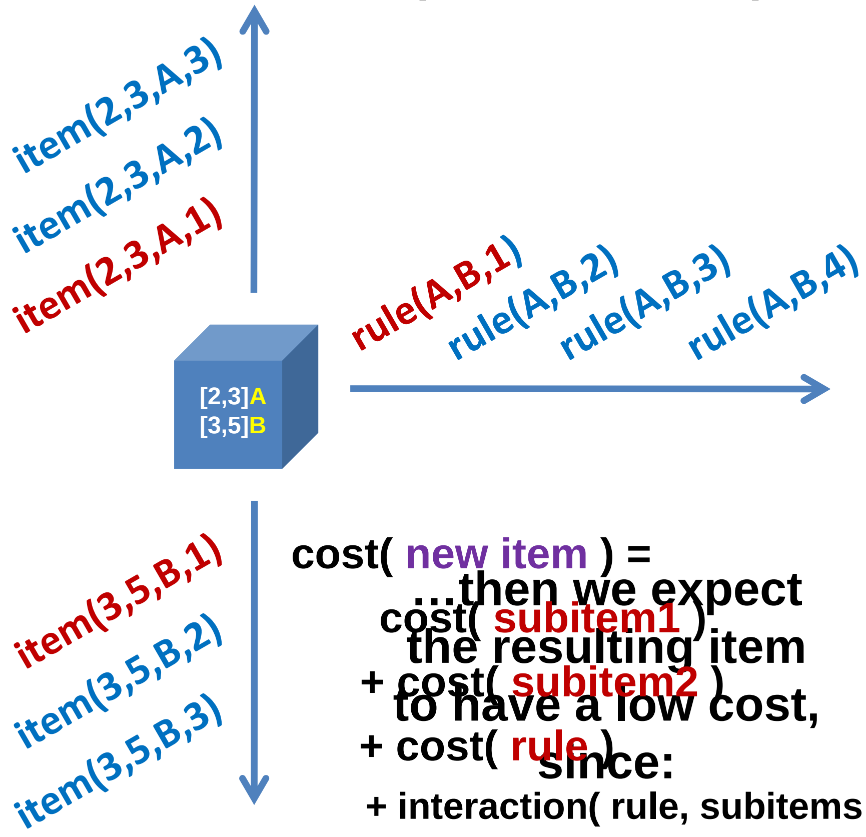
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

If we take the best representative from each axis (i.e. the “1-1-1”)…

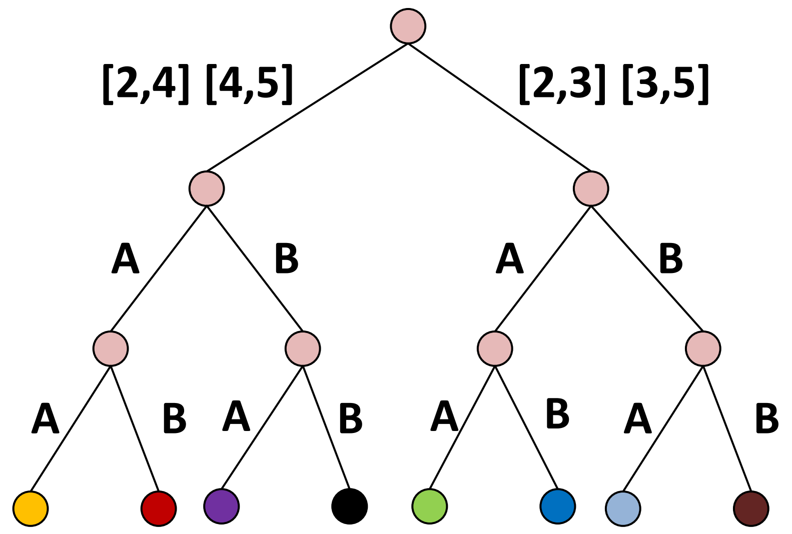


…then we expect the resulting item to have a low cost, since:

cost(new item) =
cost(subitem1)
+ cost(subitem2)
+ cost(rule)
+ interaction(rule, subitems)

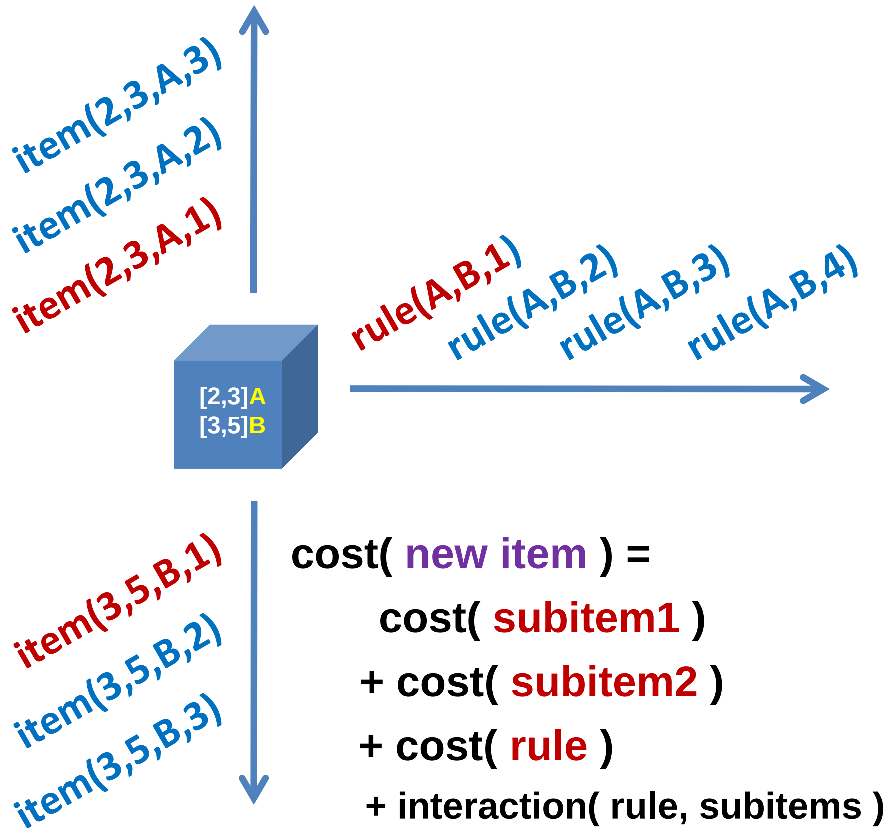
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

Though we are not guaranteed this, because:

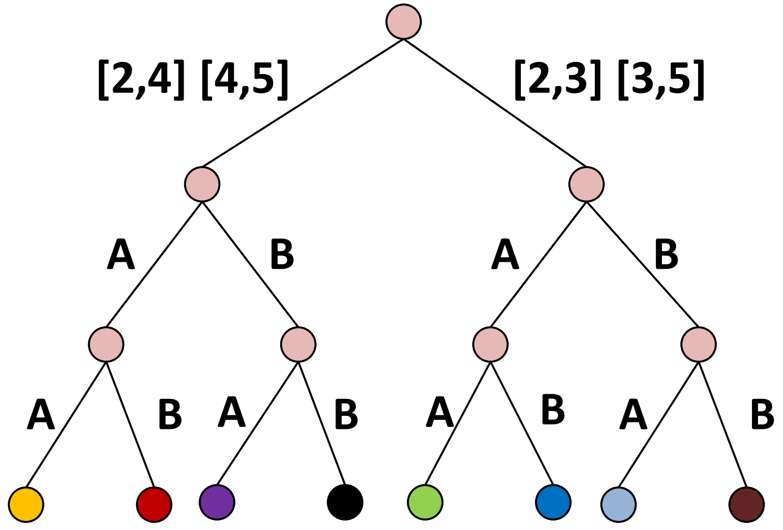


cost(new item) =
cost(subitem1)
+ cost(subitem2)
+ cost(rule)
+ interaction(rule, subitems)

this cost is not monotonic

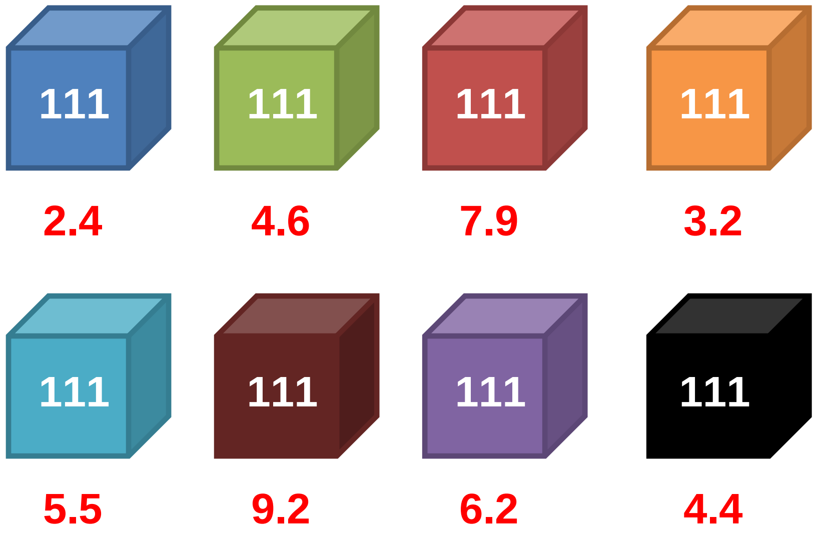
Cube pruning

A* search visits nodes in order of increasing heuristic value



A* search

Cube pruning proceeds by creating the 1-1-1 item of every cube

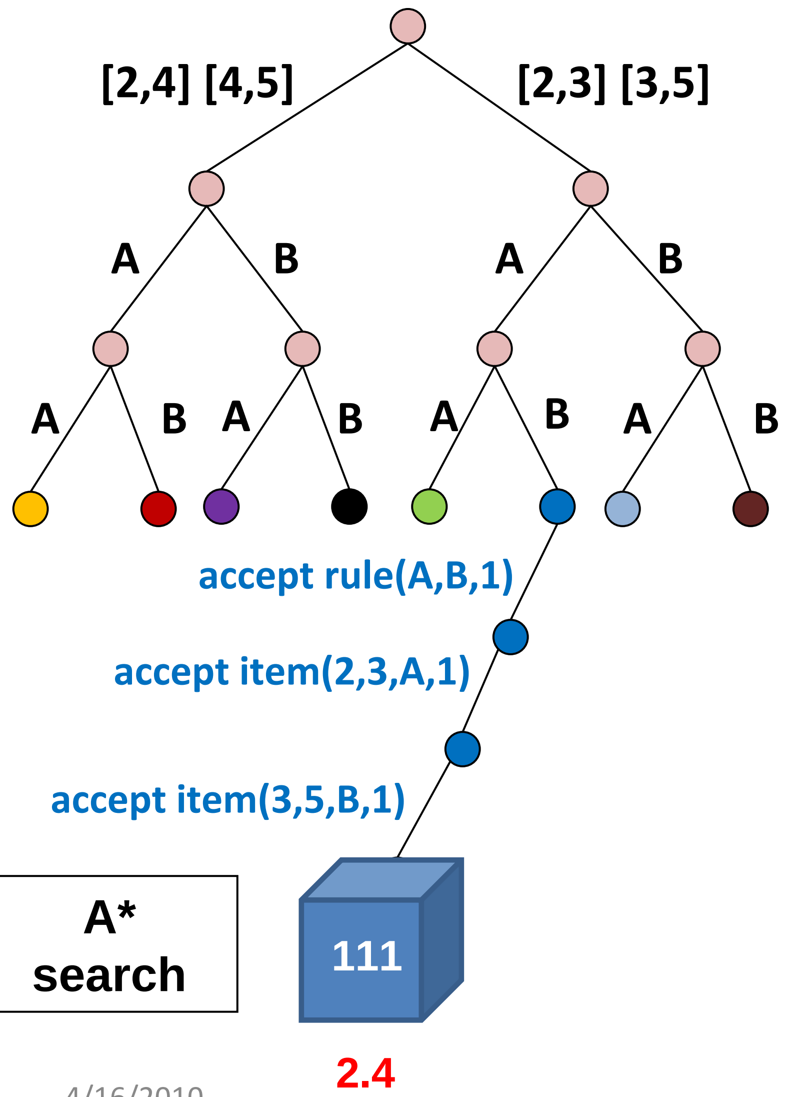


and scoring them

Cube pruning

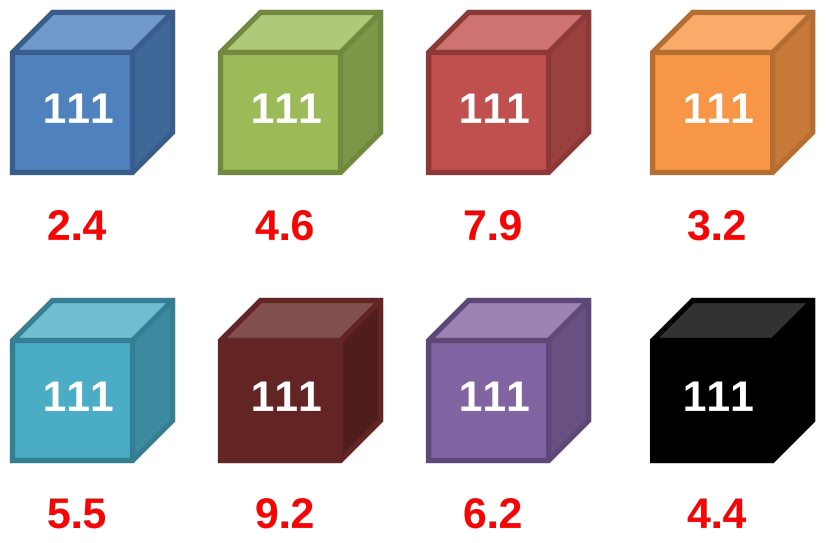
LANGUAGE WEAVER

Meanwhile, A* search has scored its frontier nodes



A* search

Cube pruning proceeds by creating the 1-1-1 item of every cube

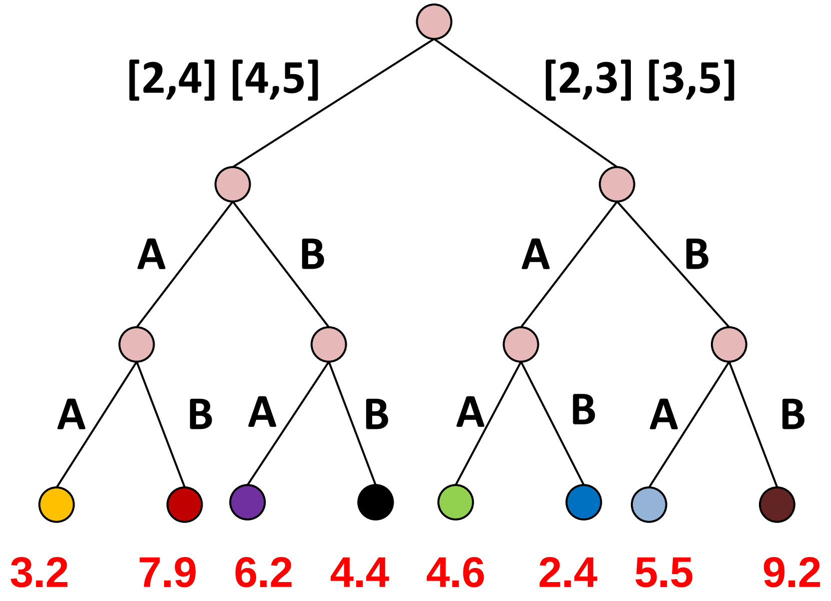


and scoring them

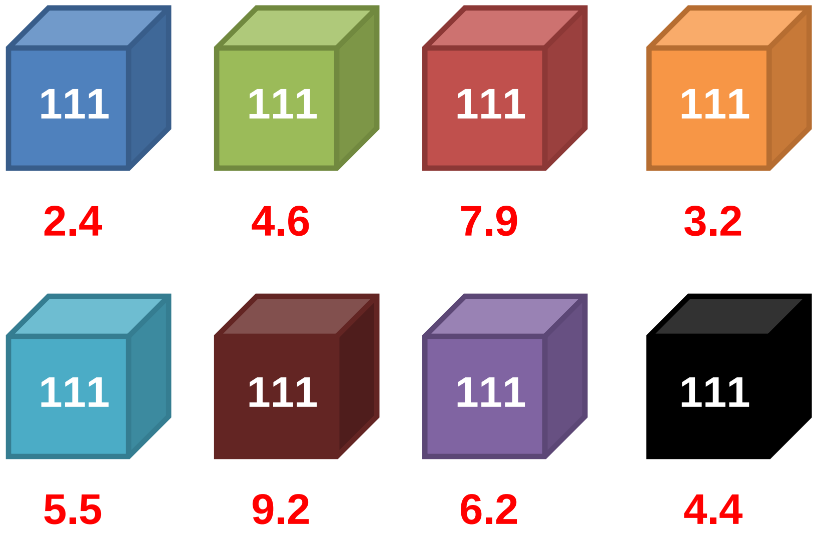
Cube pruning

LANGUAGE WEAVER

Meanwhile, A* search has scored its frontier nodes



Cube pruning proceeds by creating the 1-1-1 item of every cube

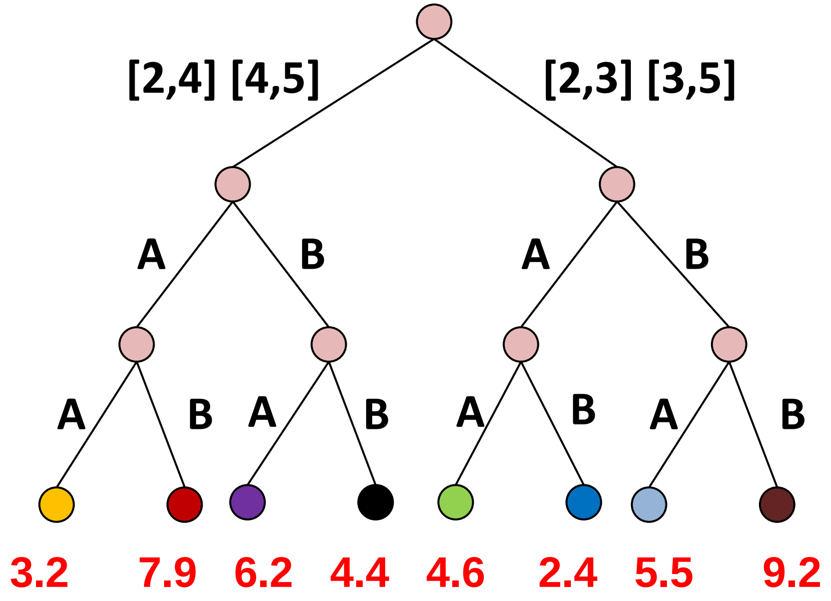


and scoring them

A* search

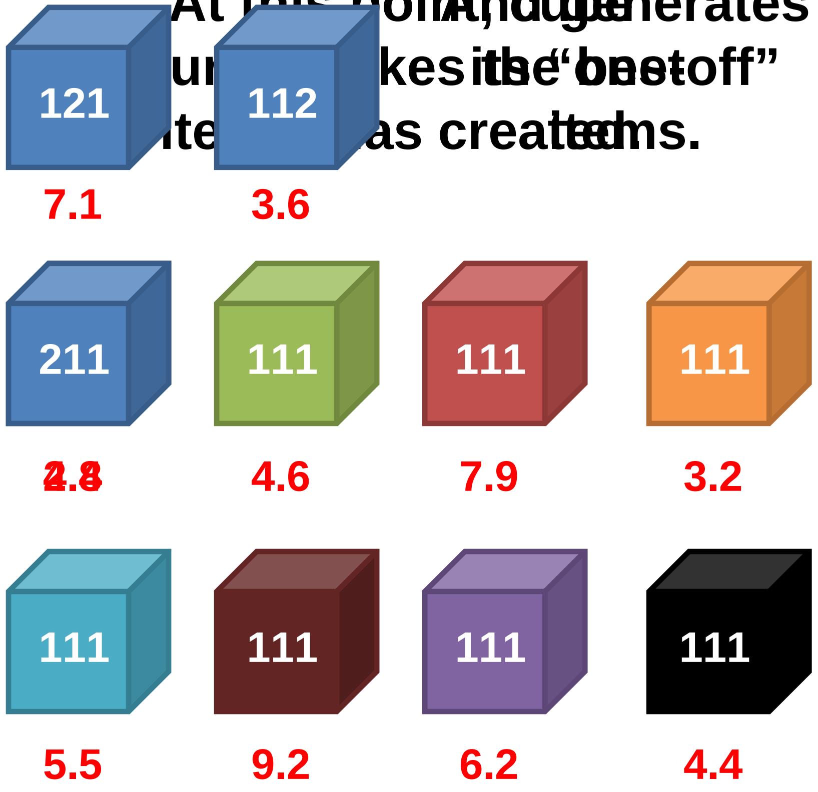
Cube pruning

Meanwhile, A* search has scored its frontier nodes



At this point, ur kes the best ite as created.

And generates its “best-off” items.



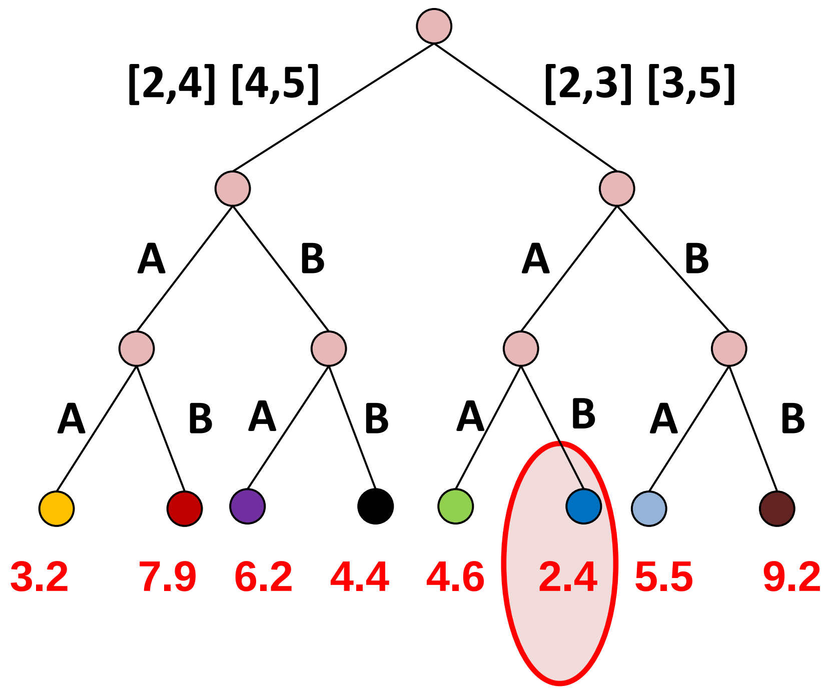
It keeps this item.

KEPT

A* search

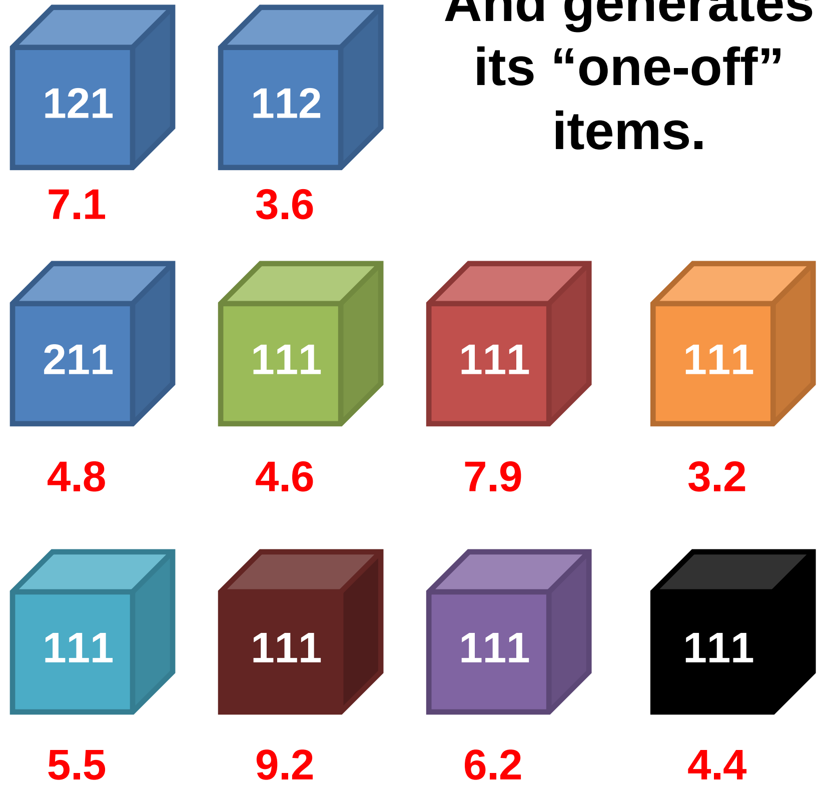
Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

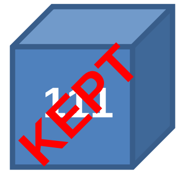


A* search

And generates its “one-off” items.

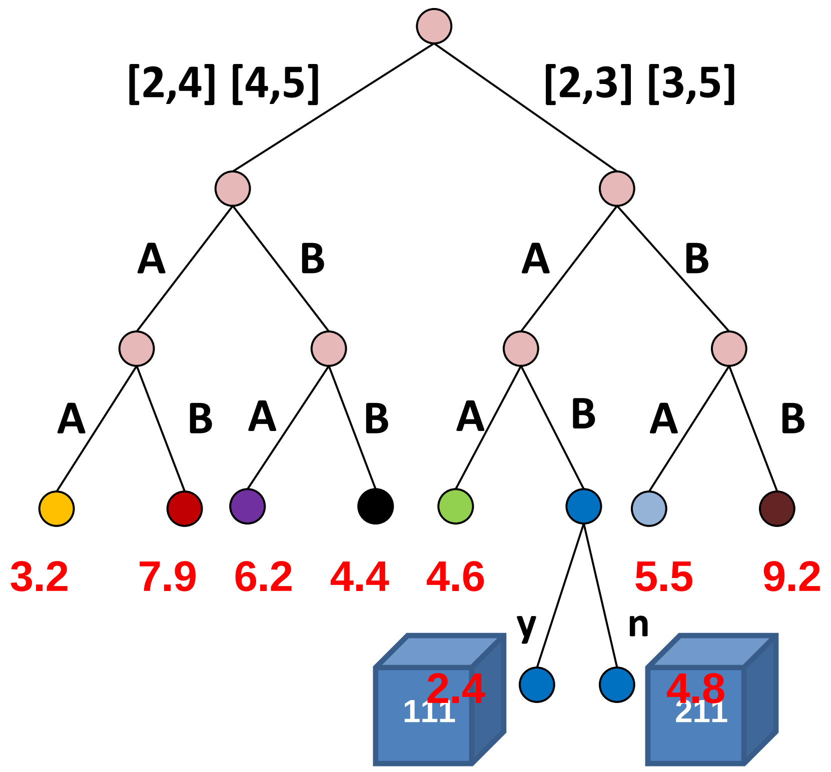


It keeps this item.



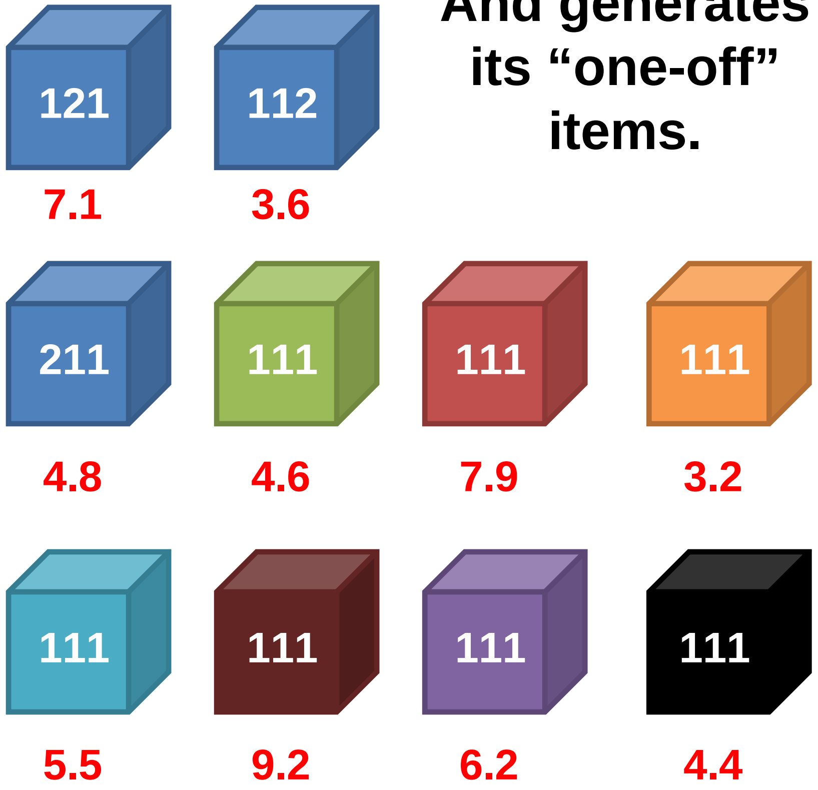
Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

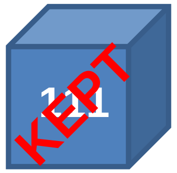


A* search

And generates its "one-off" items.

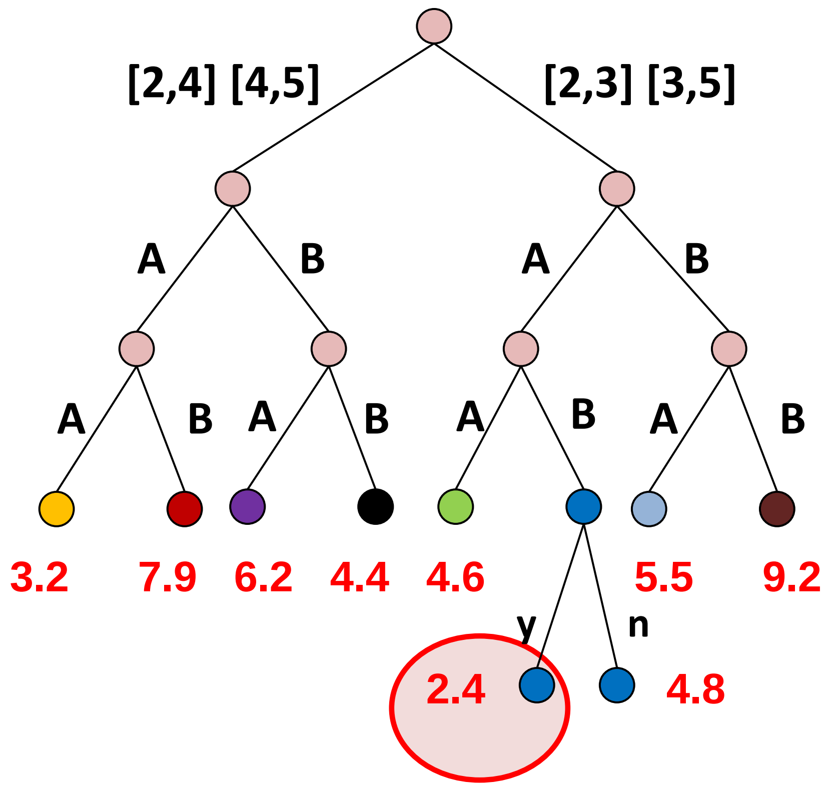


It keeps this item.



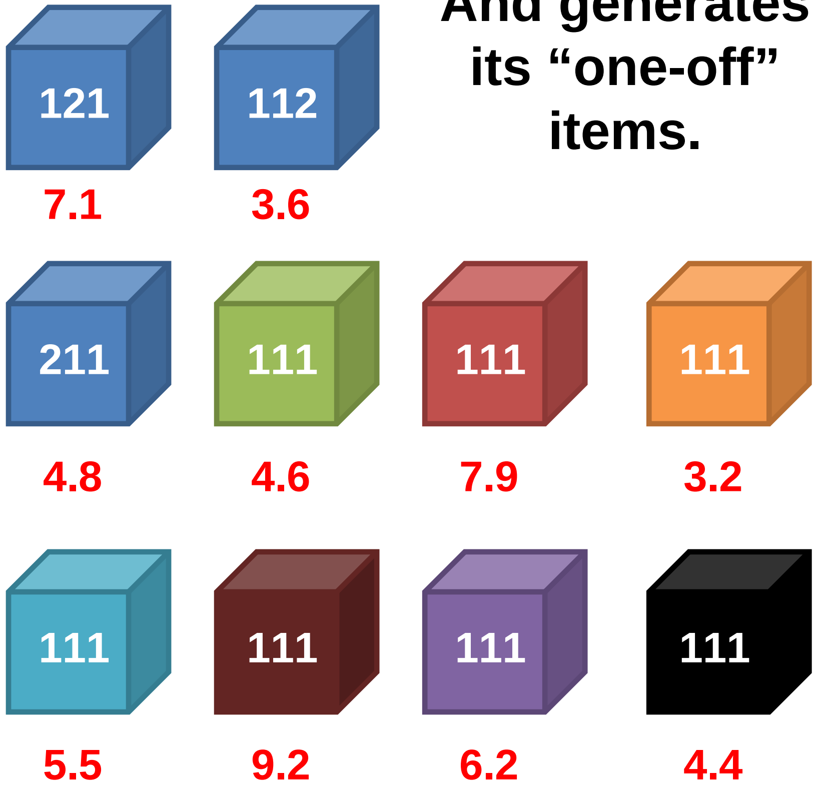
Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

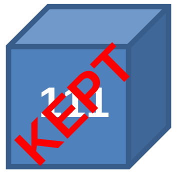


A* search

And generates its "one-off" items.

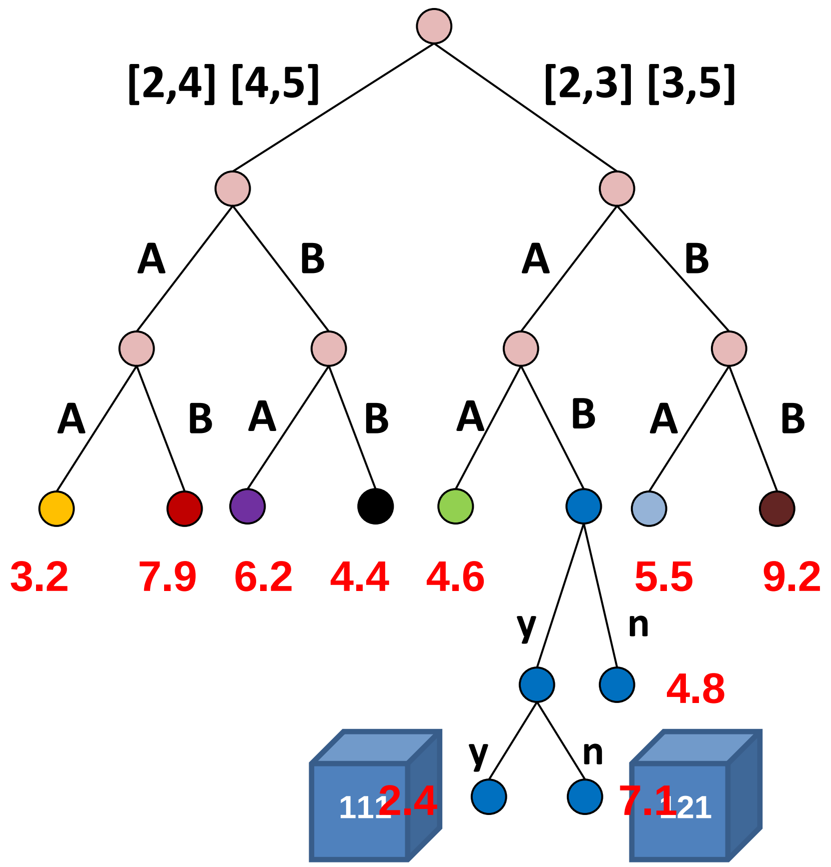


It keeps this item.



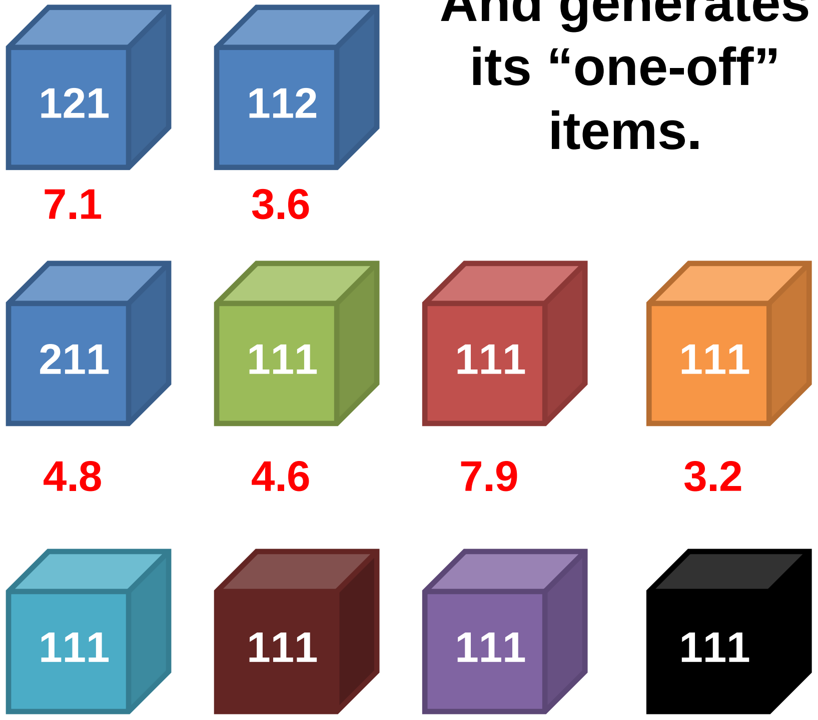
Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

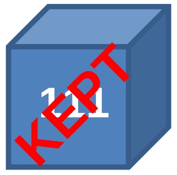


A* search

And generates its “one-off” items.

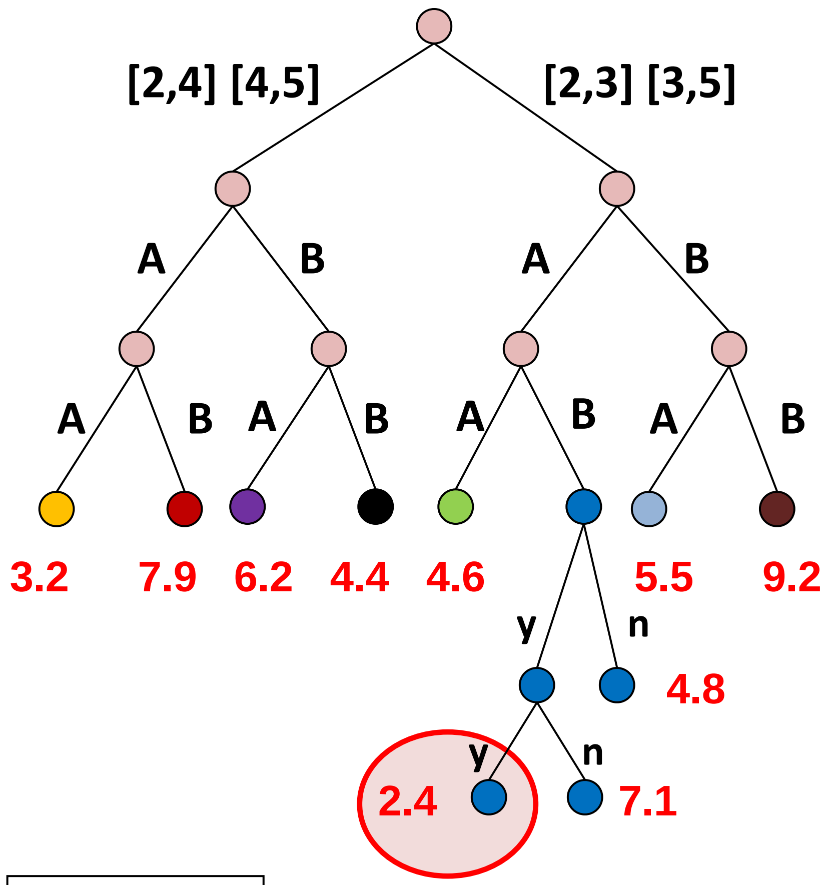


It keeps this item.



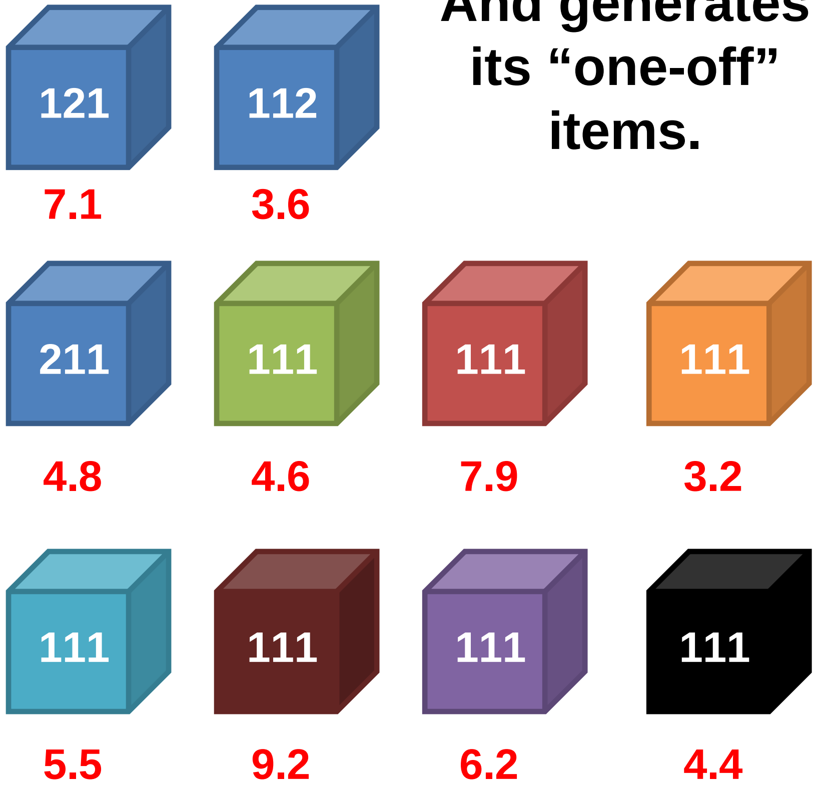
Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

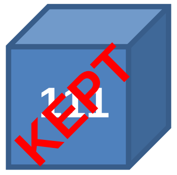


A* search

And generates its "one-off" items.

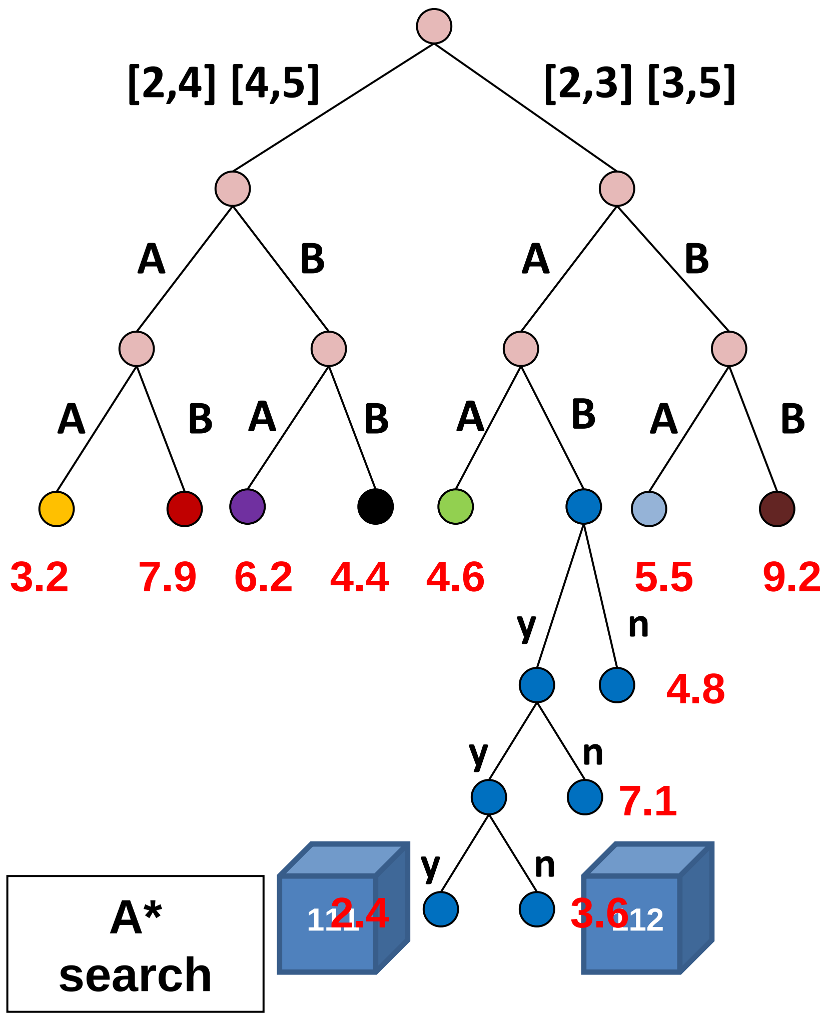


It keeps this item.

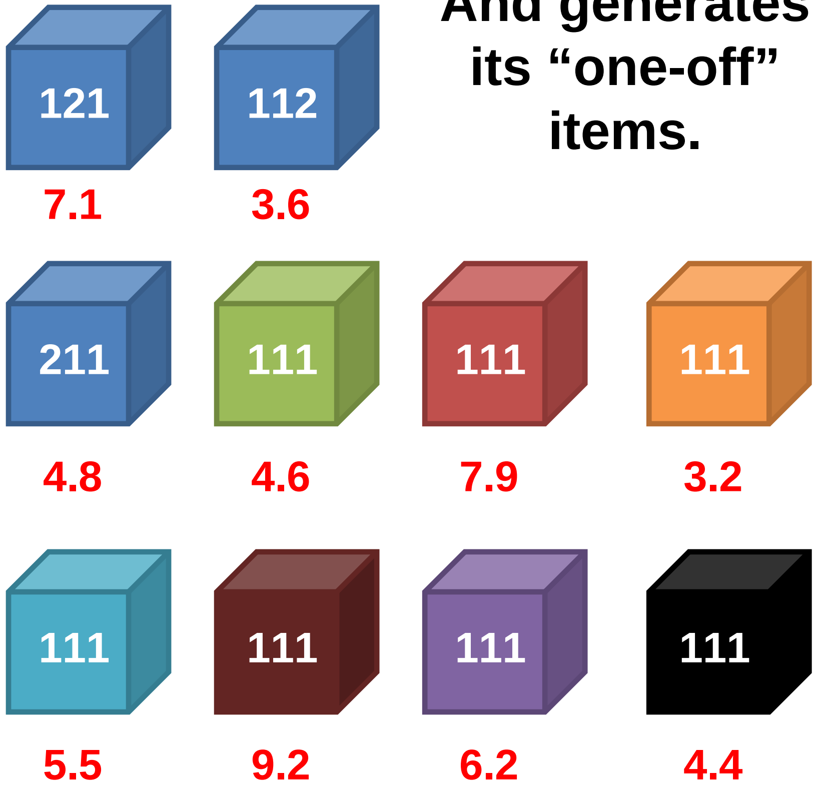


Cube pruning

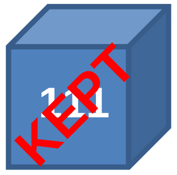
A* search continues to visit nodes in increasing order of heuristic value



And generates its "one-off" items.



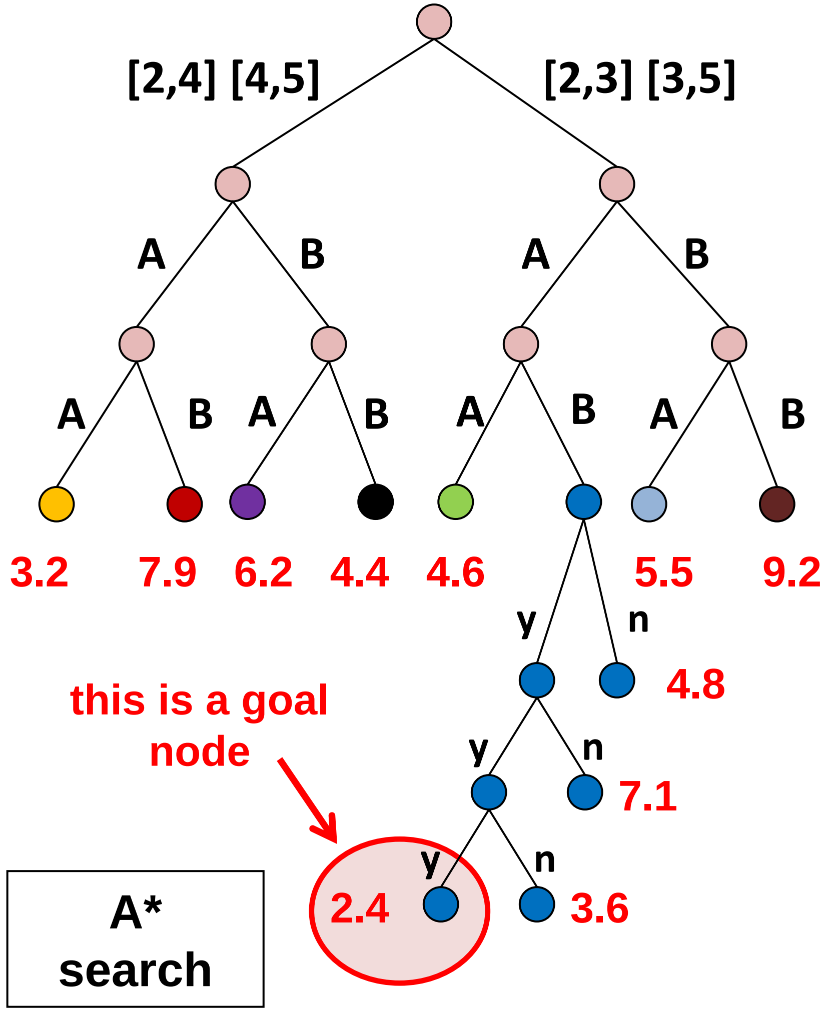
It keeps this item.



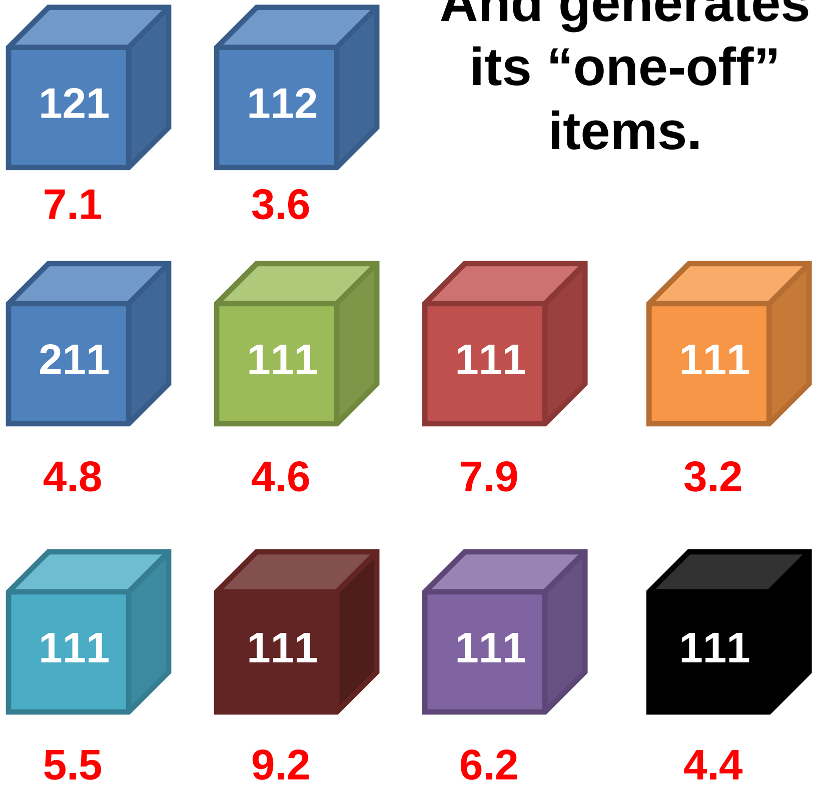
Cube pruning

LANGUAGE WEAVER

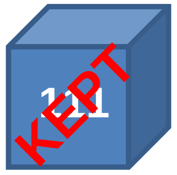
A* search continues to visit nodes in increasing order of heuristic value



And generates its "one-off" items.

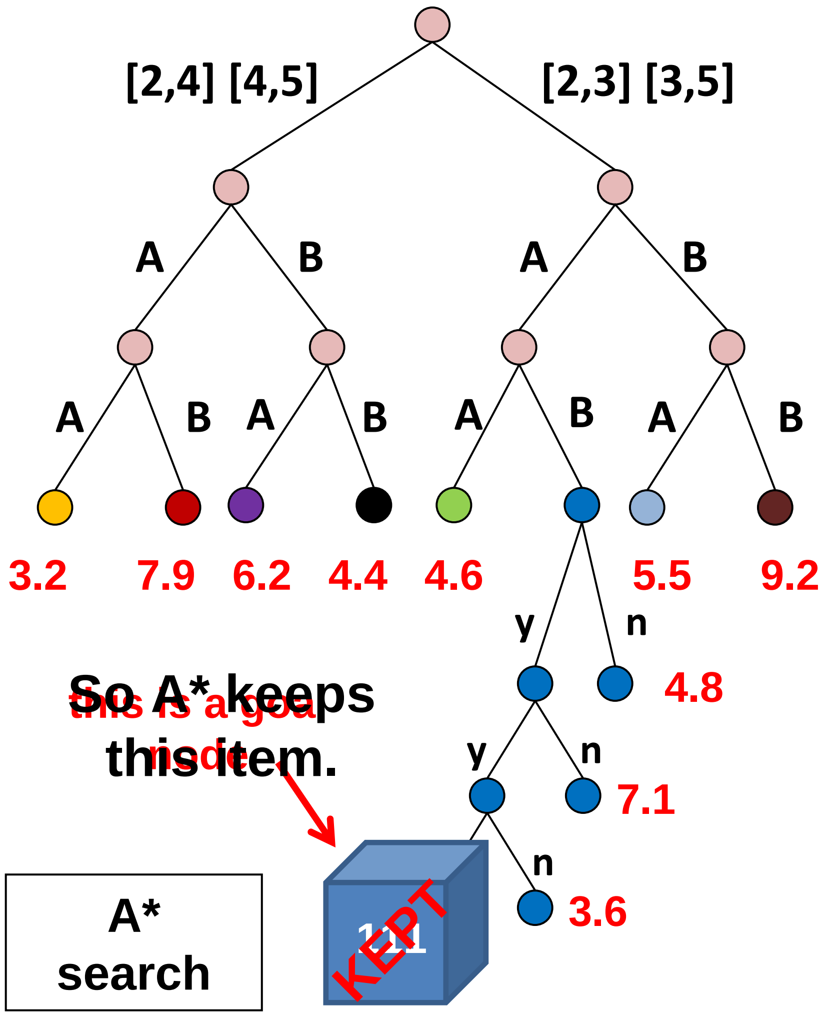


It keeps this item.



Cube pruning

A* search continues to visit nodes in increasing order of heuristic value

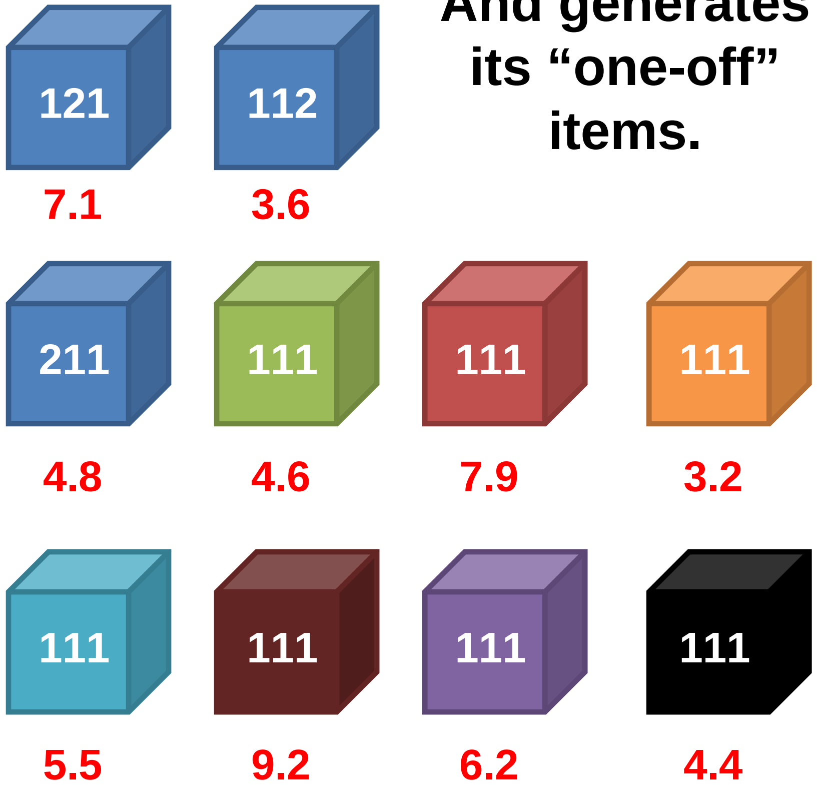


this is a goal node.

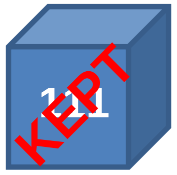
So A* keeps this item.

A* search

And generates its “one-off” items.



It keeps this item.



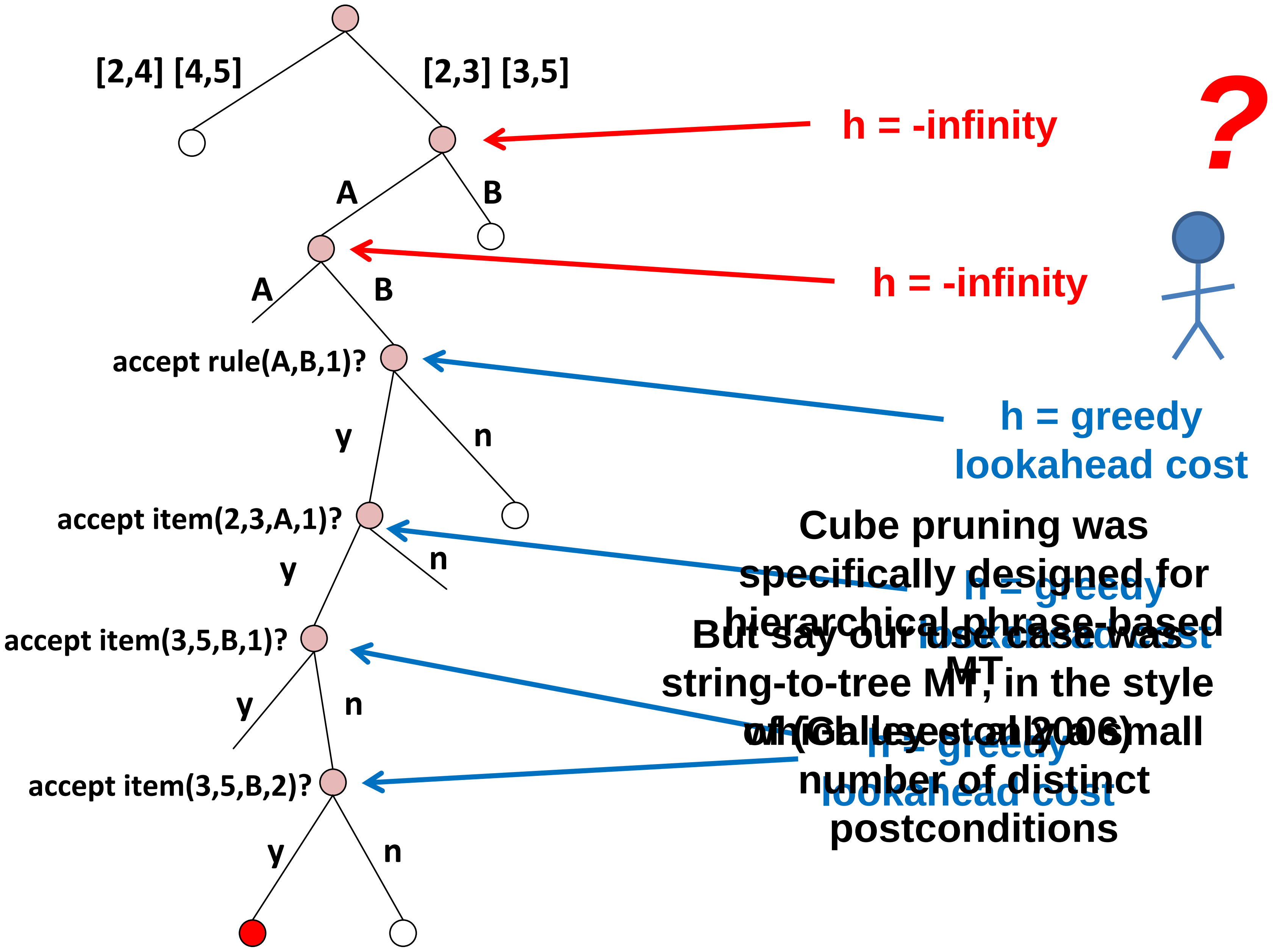
Cube pruning

LANGUAGE WEAVER

A*
search
(+ node tying)

=

Cube
pruning



Cube pruning was specifically designed for hierarchical phrase-based MT

But say our use case was string-to-tree MT, in the style of (Galley et al 2006)

which yields a small number of distinct postconditions

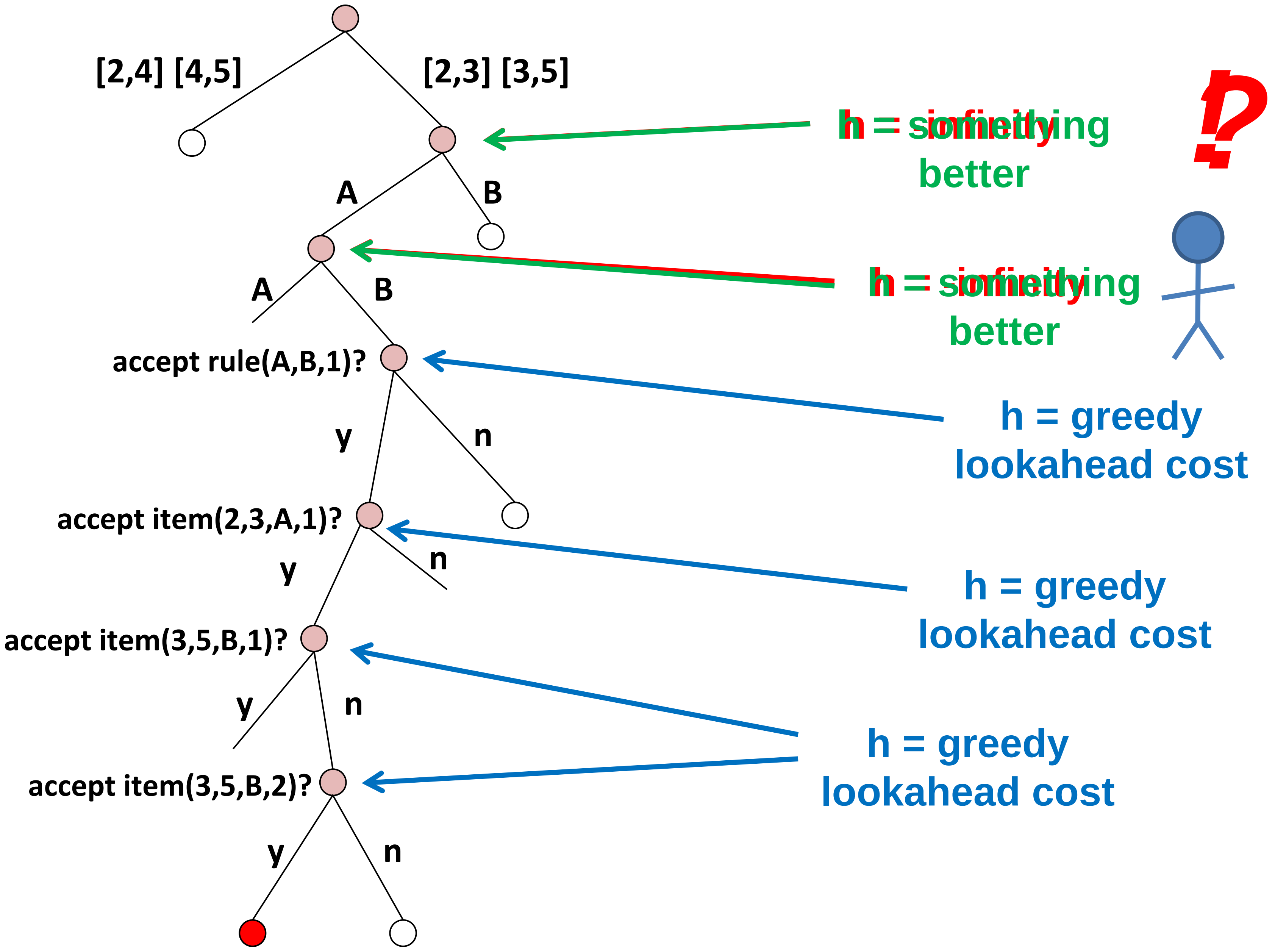
Average number of search nodes visited, per sentence

Arabic-English NIST 2008

Nodes by Type	Cube Pruning
subspan	12936
precondition	**851458**
rule	33734
item	119703
goal	74618
TOTAL	**1092449**
BLEU	**38.33**

the early nodes with infinite heuristics dominate the search time

LANGUAGE WEAVER



LANGUAGE WEAVER

Number of search nodes visited

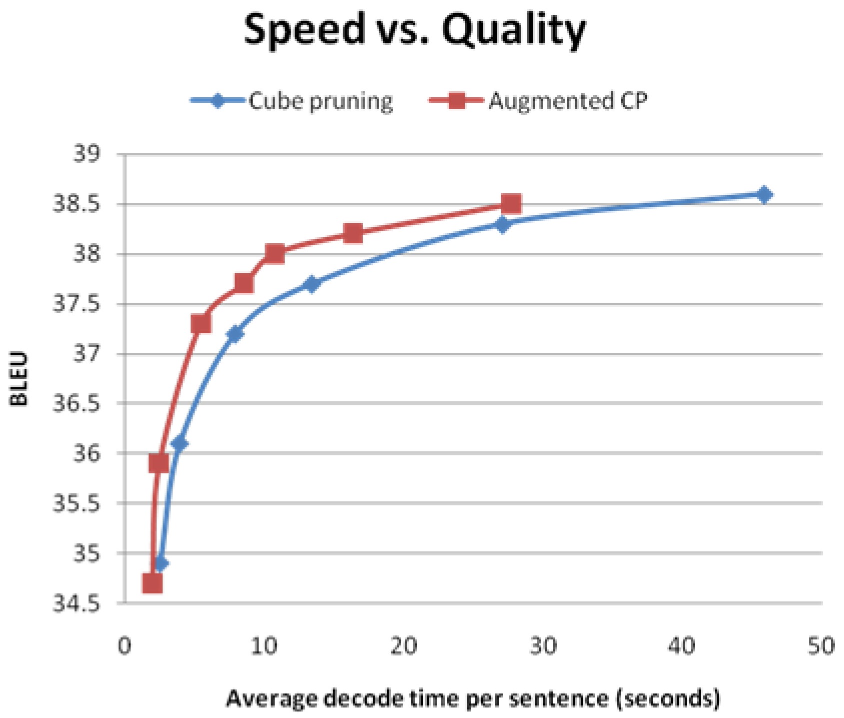
Arabic-English NIST 2008

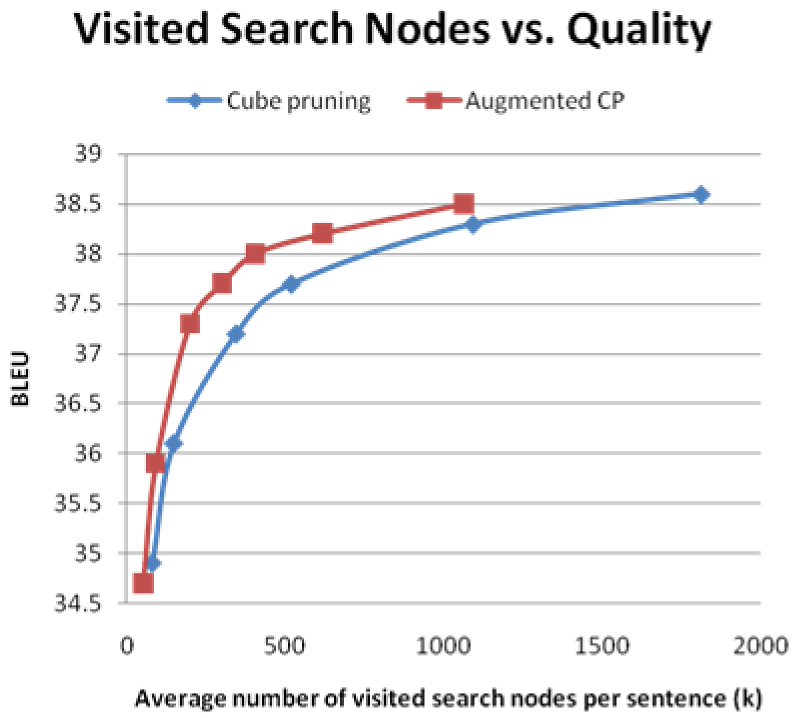
Nodes by Type	Cube Pruning	Augmented CP
subspan	12936	12792
precondition	**851458**	**379954**
rule	33734	33331
item	119703	118889
goal	74618	74159
TOTAL	**1092449**	**619125**
BLEU	**38.33**	**38.22**

LANGUAGE WEAVER

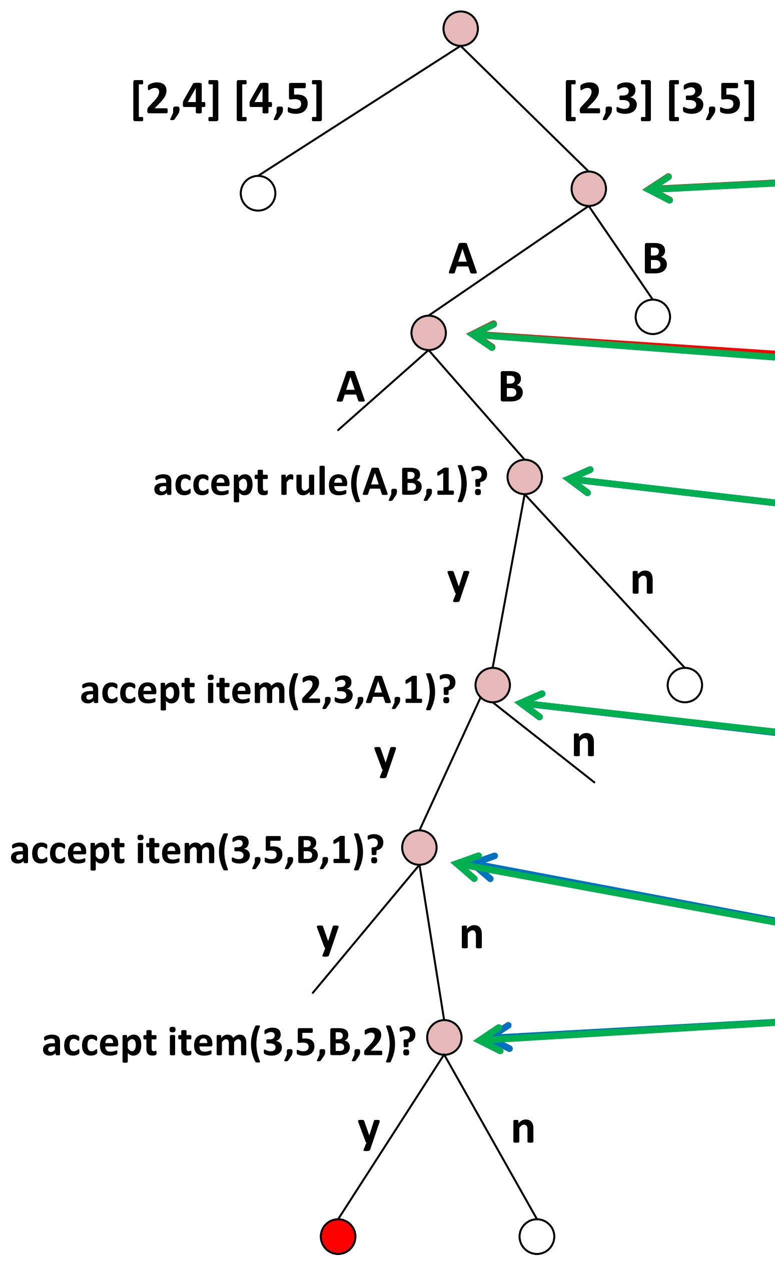
Tradeoff curves

Arabic-English NIST 2008





LANGUAGE WEAVER



h = infinity

h = admissible

h = infinity

h = admissible

Cube pruning becomes **exact**.

h = greedy

h = admissible

lookahead cost

We found that our exact version of cube pruning had a similar time/quality curve to the original inexact version of cube pruning.

This is a bit depressing, but somewhat predictable at this point.

h = greedy

h = admissible

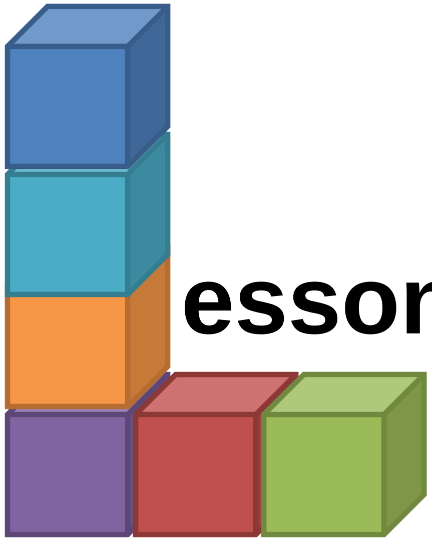
lookahead cost

However it was not as effective as our "augmented" version of cube pruning.

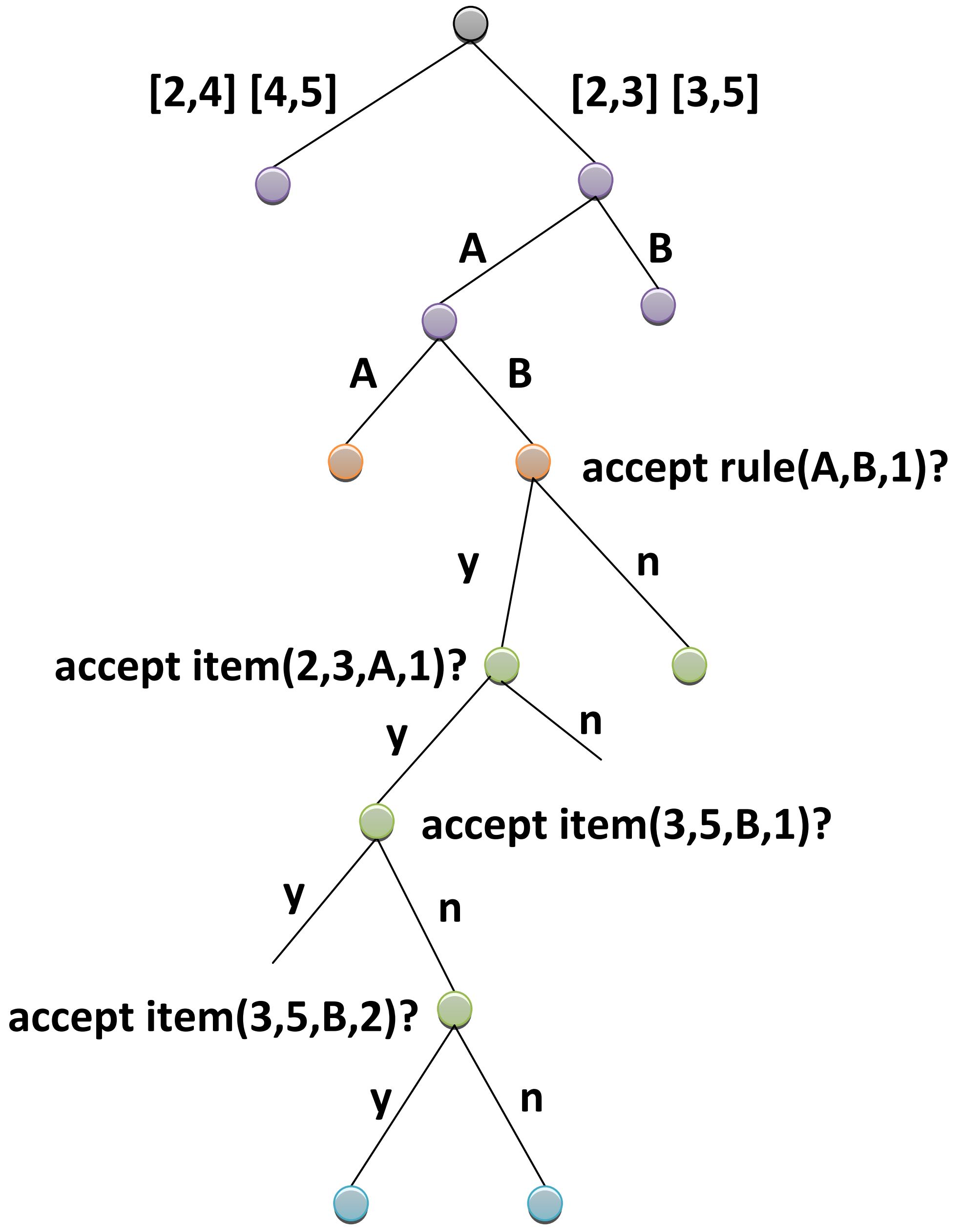
h = greedy

h = admissible

lookahead cost

What  esson

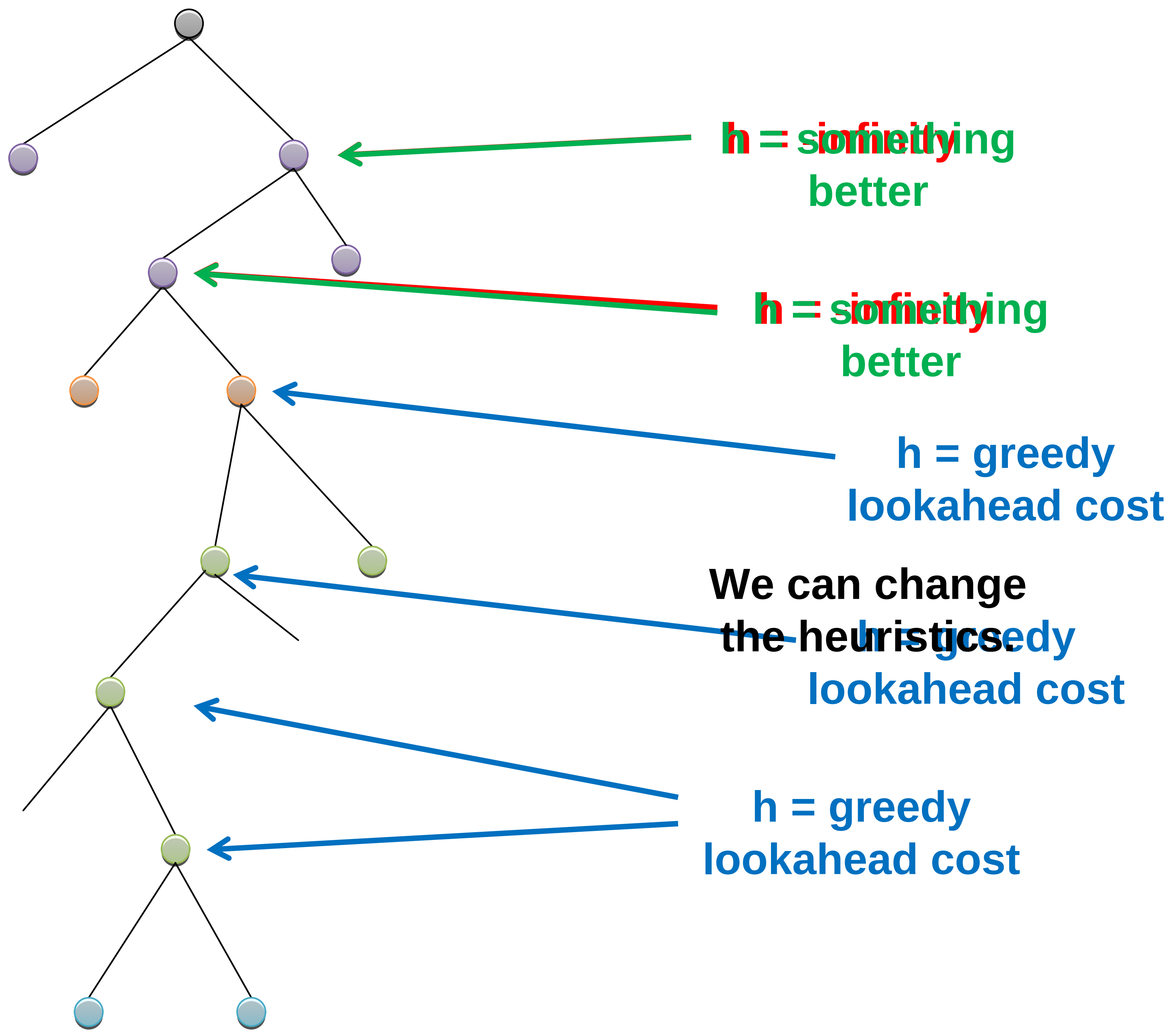
does cube pruning teach us?



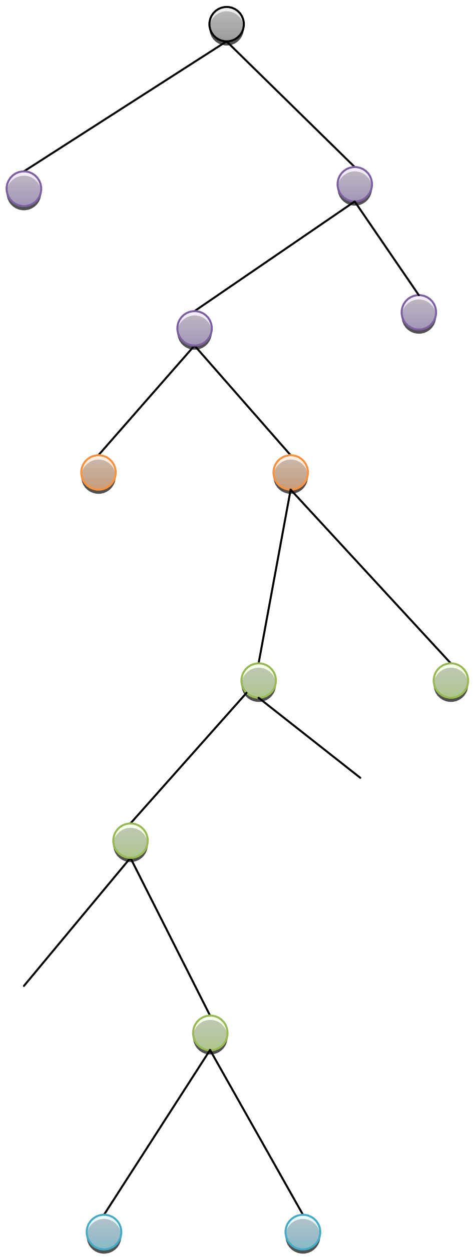
this is a search space

It tells us that it is useful to frame the CYK Item Generation Problem as a heuristic search problem.

Once this realization is made, we suddenly have **many more** avenues available to us, when implementing a CKY decoder for a particular use case


h = infinity
h = something better
h = infinity
h = something better
h = greedy lookahead cost
h = greedy lookahead cost
h = greedy lookahead cost

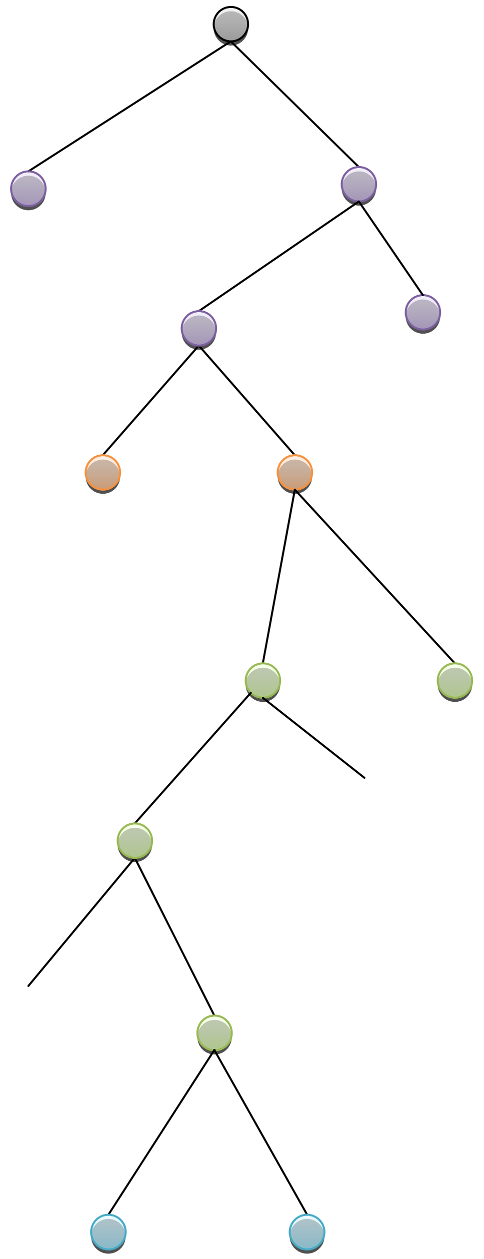
We can change the heuristics.



We can change the search algorithm.

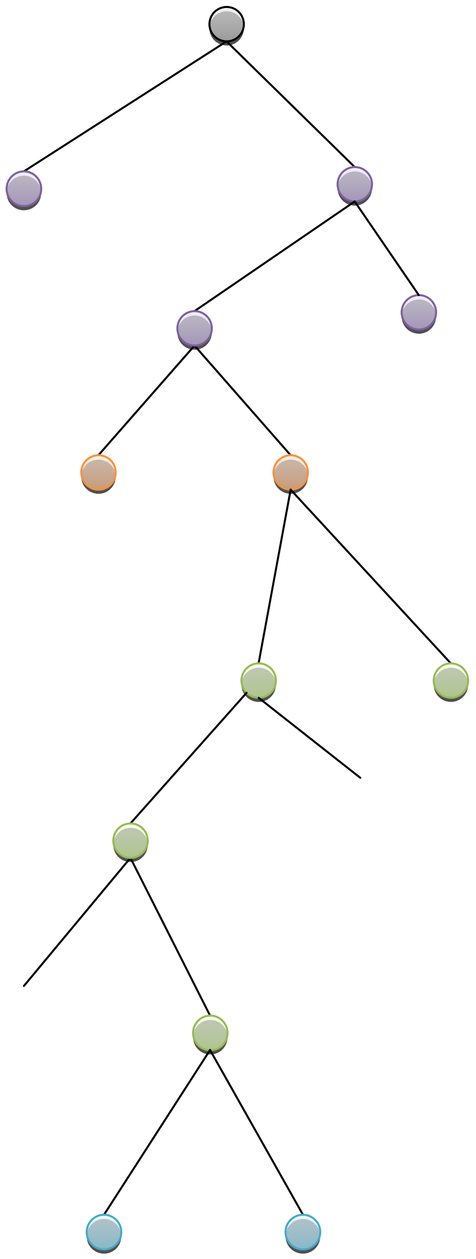
the heuristic.

LANGUAGE WEAVER



For instance, instead of A*…

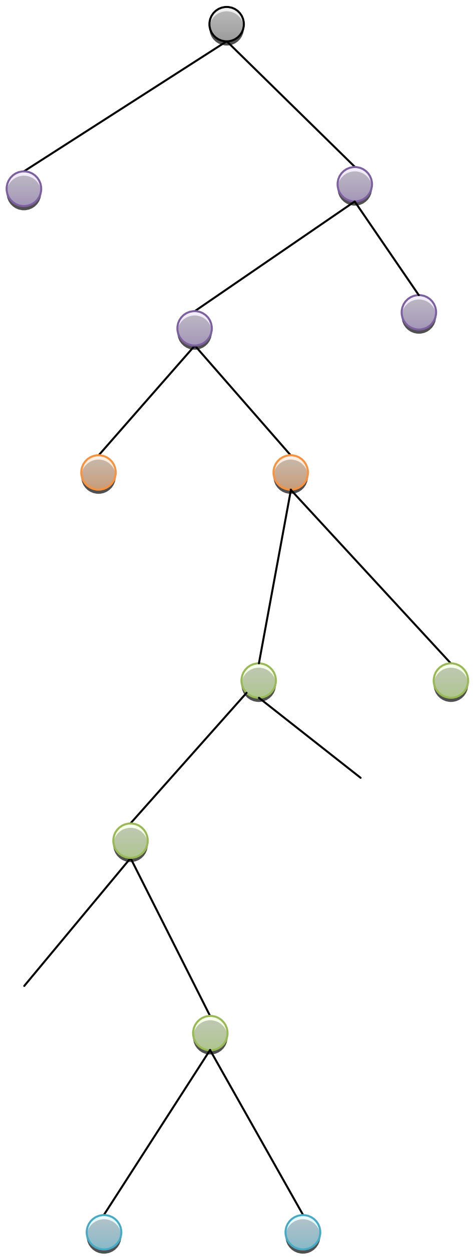
We can change the search algorithm.



For instance, instead of A*…

…we could try a depth-first strategy like depth-first branch-and-bound, and take advantage of its anytime properties.

We can change the search algorithm.

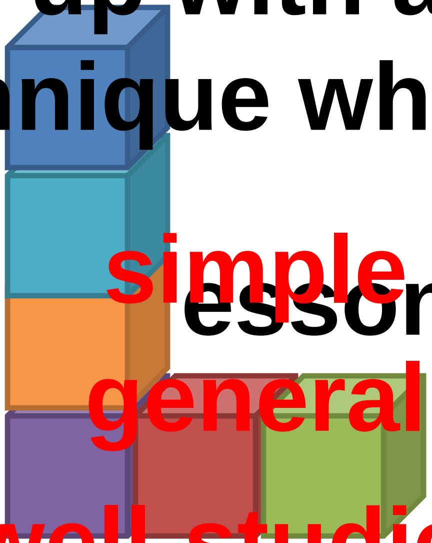


We can change

the search algorithm.

the search space.

We end up with a speedup technique which is:

What  esson

does cube pruning teach us?

simple

general

well-studied

easily adaptable to new use cases

Thank you.

Questions?